НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ім. ІГОРЯ СІКОРСЬКОГО»
Теплоенергетичний факультет
Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено:
Новоселов С.Є.
Завідувач кафедри
_____ Олександр Коваль
« _»__2020 р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інформаційні технології моніторингу довкілля»
спеціальності 122 «Комп'ютерні науки та інформаційні технології»
на тему: «Клієнт-серверне програмне забезпечення для шифрування та зберігання даних для входу на web-ресурси на базі операційної системи Linux»

Виконав (-ла):
студент (-ка) IV курсу, групи ТМ-61
Новоселов Сергій Євгенійович _____
Керівник:
Професор кафедри АПЕПС, доктор технічних наук,
Отрох Сергій Іванович _____
Рецензент:
Доцент, кандидат технічних наук, Доцент кафедри ТК
Зенів Ірина Онуфріївна _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2020 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший, бакалаврський

Напрям підготовки 6.050101 “Комп’ютерні науки”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2020 р.

ЗАВДАННЯ

на дипломну роботу студенту

_____Новоселову Сергію Євгенійовичу_____

(прізвище, ім’я, по батькові)

1. Тема роботи Клієнт-серверне програмне забезпечення для шифрування та зберігання даних для входу на web-ресурси на базі операційної системи Linux.
керівник роботи_ д.т.н. проф. Отрох С.І_____
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від _____ р. № _____

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи ___мова програмування Python, середовище розробки PyCharm, система керування базами даних SQLite_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)___Аналіз задачі розробки програмного забезпечення для конфіденційного зберігання та передавання даних. Аналіз та оцінка існуючих рішень. Вибір програмних засобів реалізації програмного рішення. Розробка програмного забезпечення для реалізації безпечного зберігання та передавання даних.

5. Перелік графічного матеріалу схема класів та БД в програмному комплексі; схема процесу шифрування та передачі інформації; графічний інтерфейс комплексу.

6. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки

АНОТАЦІЯ

Метою роботи є дослідження існуючих методів та реалізацій безпечного зберігання та передавання конфіденційних даних, а саме – даних для входу на web-ресурси. Практичне оволодіння прийомами та навичками створення клієнт-серверних програмних систем та проектів, розробка парольного менеджеру / записника з командним доступом. Мова розробки – Python. Середовище – PyCharm.

Записка містить 79 сторінок, 33 рисунків , 3 додатка і 16 бібліографічних найменувань за «Переліком посилань».

Ключові слова: Парольний менеджер, хмарне сховище, симетрична криптографія, асиметрична криптографія, Python, клієнт-сервер.

ABSTRACT

The aim of the work is to study the existing methods and implementations of secure storage and transmission of confidential data namely - credentials. Practical mastering of techniques and skills for creating client-server software systems and projects, development of a password manager / notepad storage with team access. Programming language – Python. IDE- PyCharm.

The note contains 79 pages, 33 figures, 3 appendices and 16 bibliographic names according to the "List of references".

Keywords: Password manager, cloud storage, symmetric cryptography, asymmetric cryptography, Python, client-server.

Зміст

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

WYSIWYG (What You See Is What You Get) - це система, де програмне забезпечення для редагування дозволяє редагувати вміст у формі, що нагадує його зовнішній вигляд під час друку або відображення у вигляді готового продукту.

REST API - Representational state transfer (REST) - це архітектурний стиль програмного забезпечення, який визначає набір обмежень, які будуть використані для створення веб-служб.

Майстер ключ – універсальній ключ для розшифрування усіх записів

Фреймворк - абстракція, в якій програмне забезпечення, що забезпечує загальну функціональність, може вибірково змінюватися додатковим написаним користувачем кодом, забезпечуючи таким чином програмне забезпечення, що відповідає додатку.

SSL - (англ. Secure Sockets Layer - рівень захищених сокетів) - криптографічний протокол, який має на увазі більш безпечний зв'язок.

Вступ

Менеджери паролів[1] пропонують більшу безпеку та зручність використання паролів для доступу до онлайн-сервісів. Більша безпека в основному досягається завдяки можливості більшості програм менеджерів паролів генерувати унікальні, довгі, складні, легко змінювані паролі для всіх облікових записів в Інтернеті та безпечне зашифроване зберігання цих паролів через локальне або хмарне сховище. Більшу зручність забезпечує використання єдиного головного пароля для доступу до сховища паролів, а не намагання запам'ятати різні паролі для всіх облікових записів.

Менеджер паролів допомагає генерувати, зберігати та вводити складні паролі із зашифрованої бази даних. Менеджери паролів можуть бути виконані у формі[1]:

- онлайн-сервісів;
- локально встановленого програмного забезпечення;
- апаратних пристроїв.

Враховуючи зростання кількості веб-ресурсів із можливістю реєстрації та збільшення ризику зламу аккаунтів, використання менеджерів паролів стає здебільше актуальним. Більш того, більшість компаній має внутрішні корпоративні ресурси, доступ до яких надає співробітникам. Для організації безпечної передачі та зберігання конфіденційної інформації, а саме - даних для входу, потрібне програмне забезпечення, яке безпечно і автоматично організує цей процес.

На ринку не існує безпечних систем[2], які дозволяють ділитися своїми даними авторизації із іншими користувачами без передачі майстер-ключу, який надає доступ до інформації, яка зберігається у базі даних. Компрометація ключа компрометує будь-які майбутні його застосування. Однак, навіть якщо факт компрометації ключа буде швидко виявлено та вжито заходів з його анулювання, то під загрозою опиняються колишні його застосування. Наприклад, зловмисник може мати отриманий цим ключем шифротекст і тепер у нього з'явиться можливість його дешифрувати.

Ця записка включає в себе наступні розділи, пов’язані із розробкою програмного забезпечення:

- **Постановка задачі**

Цей розділ включає в себе опис призначення програмного забезпечення та його потенційних користувачів.

- **Опис предметної області**

Аналіз аналогічних комерційних та безкоштовних продуктів, їх переваги та недоліки.

- **Засоби розробки**

Опис мови програмування, сторонніх бібліотек та програмних середовищ шифрування які були використані у процесі розробки.

- **Опис програмної реалізації**

Опис методів шифрування та авторизації. Концептуальна модель бази даних, діаграма класів та use-case діаграма. Перелік реалізованих API-методів.

- **Робота користувача з програмною системою**

Системні вимоги та інсталяція. Сценарій роботи користувача з системою. Скріншоти роботи програми.

1. Постановка задачі

1.1 Призначення програмного продукту

З причини поширення кібератак на злом пароля[3] користувачам доводиться використовувати складні паролі для акаунтів, що створює складність їх запам'ятовувати для різних сайтів. Виникає питання про способи зберігання паролів. Найбільш більш ефективний спосіб зберігання паролів є менеджер паролів.

Менеджери паролів можуть зберігати дані локально на комп'ютері або віддалено в «хмарі»[4]. Переваги хмарного зберігання і синхронізації очевидні: паролі доступні на всіх пристроях, на яких встановлено менеджер. Ризик полягає в тому, що якщо хмарний сервіс буде скомпрометований, паролі виявляться в руках зловмисників.

В більшості менеджерів паролів захист ключа шифрування — головна умова безпеки паролів. Якщо зловмисник дізнається цей ключ, то легко зламає всю базу паролів незалежно від того, наскільки складний алгоритм шифрування використовується. Тобто, якщо виникає необхідність передати зашифровані дані іншій людині треба повідомити майстер-ключ. Але не завжди можна бути певним в безпеці каналу передачі. Такий функціонал може бути корисний для передачі даних для входу на корпоративні ресурси, між співробітниками компанії.

1.2 Мета розробки

Метою розробки є реалізація Linux-сумісної програмної системи, що надає змогу зберігати та передавати конфіденційні дані по незахищеним каналам, а також синхронізувати дані з хмарним сховищем.

З огляду на вище зазначені проблеми встає задача розробки клієнт-серверного програмного забезпечення, яке має наступні функції:

- Безпечне зберігання даних, зашифрованих стійким алгоритмом, користувачу доступна синхронізація своїх записів із хмарним сховищем.
- Серверне ПЗ не має ключів для розшифровування даних користувачів

- Система дозволяє поділитися будь-яким своїм записом за допомогою асинхронної криптографії.
- Виділений сервер, який можна розгорнути навіть у локальній корпоративній мережі.

Такий функціонал дозволить отримати менеджер паролів, який доступний з будь-якого пристрою, але компрометація хмарного середовища не викриє конфіденційні дані.

2. Опис предметної області

Були вибрані найпопулярніші менеджери паролів, в яких присутній функціонал синхронізації[5]. Деякі з них мають функціонал командного доступу до записів.

2.1.Порівняння рішень на ринку

2.1.1. LastPass.

Рішення від розробників популярного розширення для десктопних браузерів, яке раніше займалося зберіганням і синхронізацією паролів. Мобільний додаток вміє зберігати в зашифрованому вигляді паролі та інші дані (в платній версії), синхронізувати збережені дані між різними пристроями і браузерами, і має версії для всіх основних ОС і браузерів. Захист даних здійснюється на основі шифрування AES-256, всі дані зберігаються на сервері розробників, отже, питання цілісності і доступності даних залишається відкритим. На рисунку 2.1.1.1 зображене це програмне забезпечення.

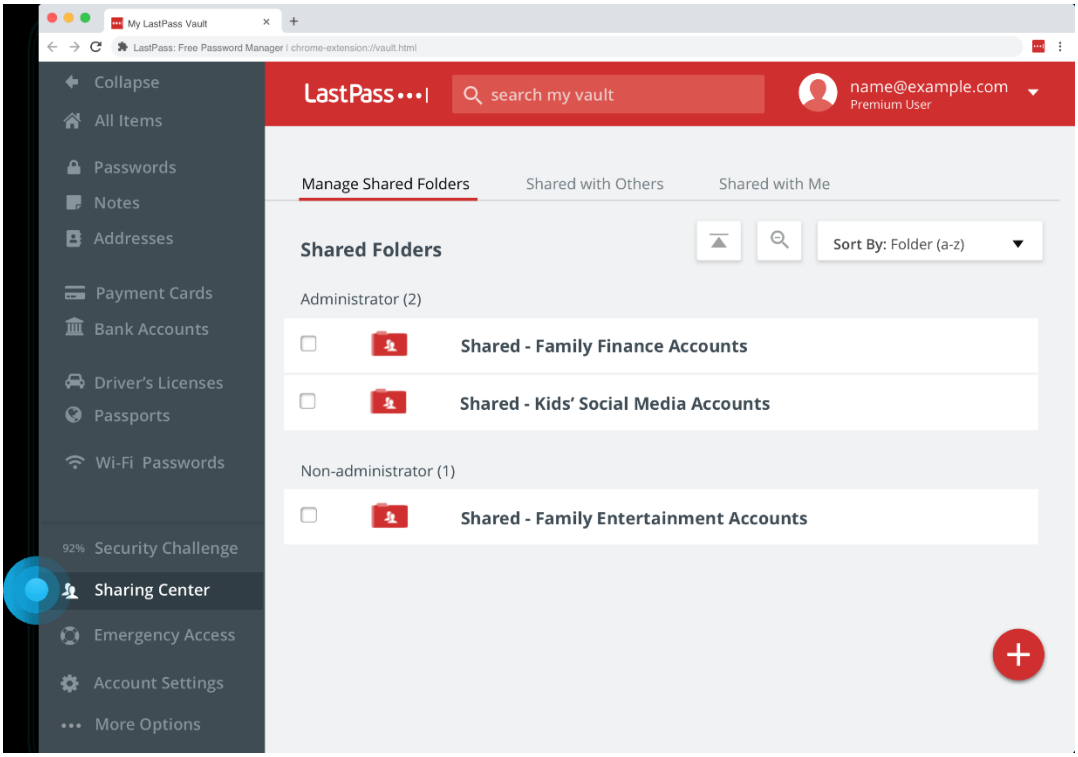

Рисунок 2.1.1.1

2.1.2. Keeper.

Менеджер паролів з великою кількістю функцій (Рисунок 2.1.1.2). Додаток охоплює основні функції, включає в себе автоматичне заповнення в різних додатках і веб-сайтах Присутня синхронізація записів між пристроями і хмарним сховищем. Також можливо зберігати дані в сховище. Функціоналу доступу до запису різних користувачів немає.

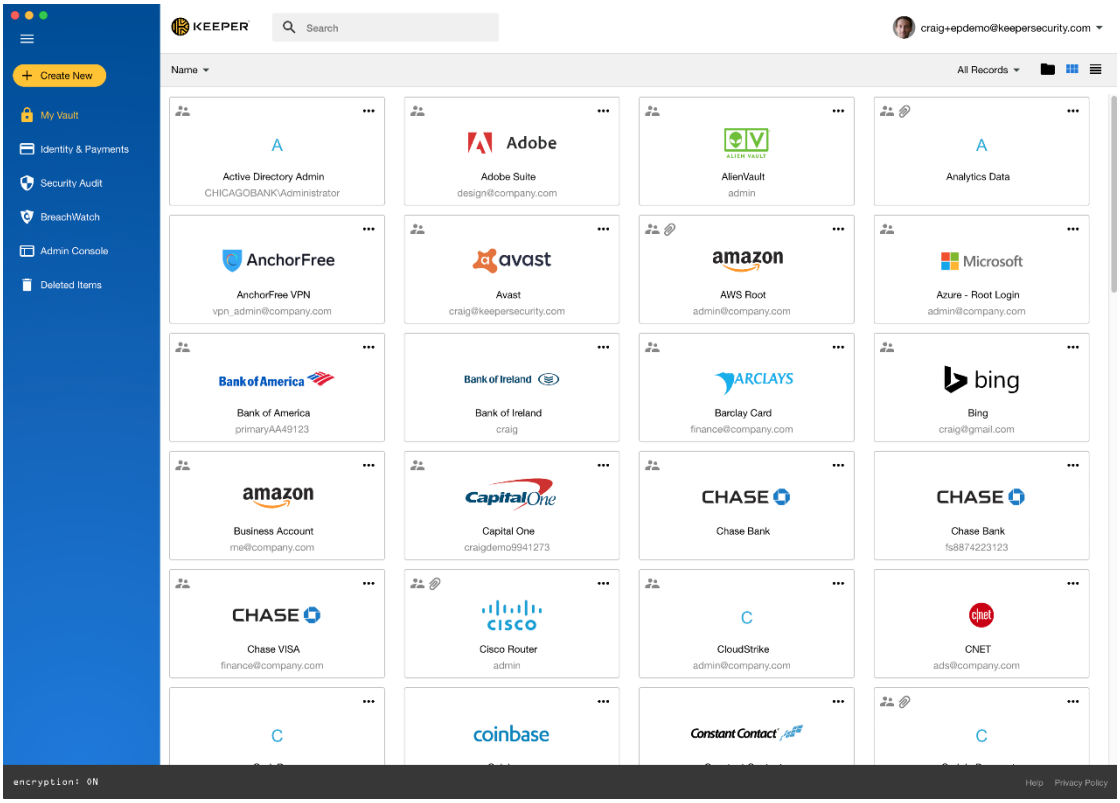

Рисунок 2.1.1.2

2.1.3. CommonKey

Менеджер паролів для командної роботи (Рисунок 2.1.1.3). Підтримка: Web, Chrome. Вартість: безкоштовно для груп до 3-х користувачів; 2$ за одного користувача в місяць. Виділеного серверу також немає, дані зберігаються на серверах розробників.

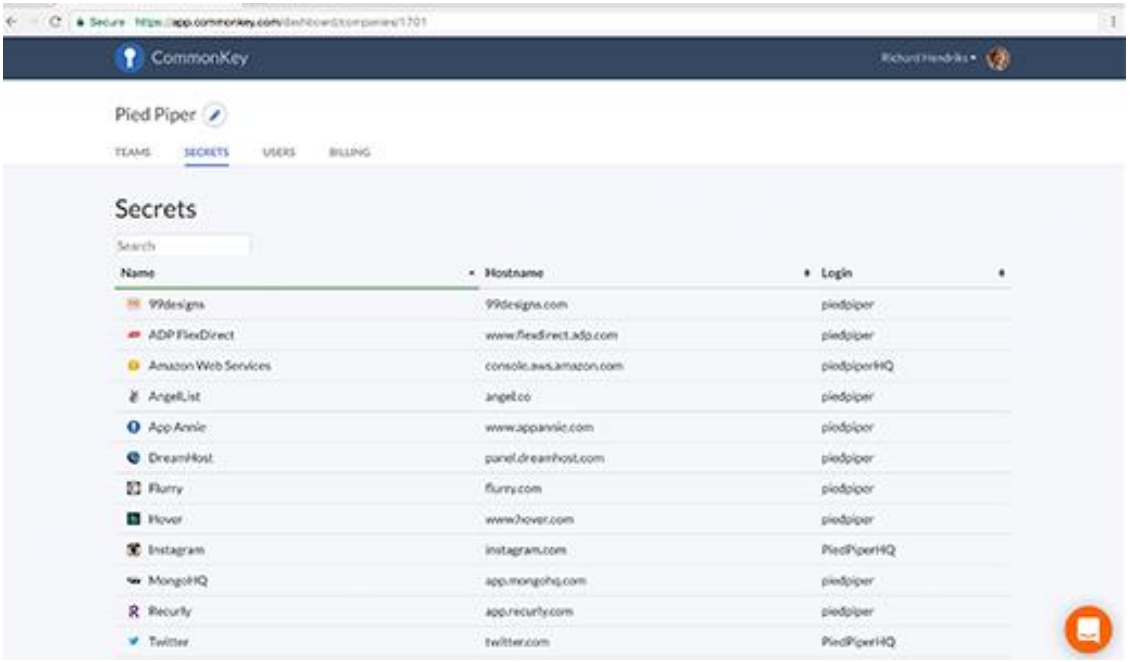

Рисунок 2.1.1.3

2.1.4 Інші

Усі інші парольні менеджери мать ідентичний функціонал, але розробники не надають можливість отримати або купити виділений сервер. Серед них: Zoho Vault, Meldium, Vaultier, SimpleSafe, Passwork

3. Засоби розробки

3.1 Python

Python — високорівнева мова програмування загального призначення, орієнтована на підвищення продуктивності розробника і легке читання коду[6]. Синтаксис ядра Python мінімалістичний. У той же час стандартна бібліотека включає великий обсяг корисних функцій.

Python підтримує структурне, об'єктно-орієнтоване, функціональне, імперативне і аспектно-орієнтоване програмування[6]. Основні архітектурні риси — динамічна типізація, автоматичне керування пам'яттю, повна інтроспекція, механізм обробки виключень, підтримка багатопоточних обчислень, високорівневі структури даних. Підтримується розбиття програм на модулі, які, в свою чергу, можуть об'єднуватися в пакети.

Еталонною реалізацією Python є інтерпретатор CPython, що підтримує більшість активно використовуваних платформ. Саме за причин кросплатофрменості ця мова була обрана для реалізації програмного продукту.

До плюсів цієї мови розробки можна віднести:

- Python легко вивчити навіть для початківців розробників. Його код легко читати. Також ви можете легко виконувати безліч складних функцій завдяки стандартній бібліотеці.
- Підтримує декілька систем і платформ.
- Об'єктно-орієнтоване програмування.
- У Python є безліч фреймворків, які роблять веб-програмування дуже гнучким.
- Покращує швидкий розвиток, використовуючи менше коду. Навіть невелика команда може ефективно впоратися з Python.
- Дозволяє легко масштабувати навіть найскладніші програми.
- Для Python доступна велика кількість ресурсів

До мінусів цієї мови розробки можна віднести:

- Python повільний
- Python - не дуже гарна мова для мобільного розвитку.
- Python - не гарний вибір для завдань, що потребують пам'яті.
- Побудувати високо графічну 3D-гру за допомогою Python майже неможливо.
- Має обмеження в доступі до бази даних.
- Python не підходить для роботи з багатопроцесорними / багатоядерними.

3.1.1 PyQT

PyQt — прив'язка Python до набору інструментів QT, реалізована як плагін Python[7]. PyQt — це вільне програмне забезпечення, розроблене британською фірмою Riverbank Computing.

PyQt реалізує близько 440 класів та понад 6000 функцій та методів , включаючи:

- великий набір віджетів GUI
- класи доступу до баз даних SQL (ODBC, MySQL, PostgreSQL, Oracle, SQLite)
- QScintilla, віджет для редагування текстів на основі Scintilla
- віджети з інформацією про дані, які автоматично заповнюються з бази даних
- аналізатор XML
- Підтримка SVG

Програмування GUI з Qt побудовано навколо концепції сигналів та слотів для зв'язку між об'єктами. Сигнал випромінюється, коли відбувається подія (наприклад, натиснута кнопка), а слоти - це виклики функції, які керують подією (наприклад, показують спливаюче вікно, коли натискається кнопка). Це дозволяє забезпечити гнучкість при обробці подій GUI і приводить до більш чистої бази коду.

PyQt має підтримку для завантаження інтерфейсів, створених за допомогою Qt Designer, конструктора інтерфейсів із перетягуванням WYSIWYG, який дозволяє проектувати та створювати інтерфейси без написання коду.

3.1.2 Flask

Flask — фреймворк для створення веб-додатків на мові програмування Python, що використовує набір інструментів Werkzeug. Відноситься до категорії так званих мікрофреймворков — мінімалістичних каркасів веб-додатків, та надає лише базові функції.

3.2 SQLite

SQLite — це вбудована база даних, яка підтримує досить повний набір команд SQL і доступна у вихідних кодах (на мові C)[8].

Враховуючи невелику кількість таблиць та задачу спрощення розгортання розробленої програми, була обрана вбудована БД SQLite.

Слово «вбудована» (embedded) означає, що SQLite не використовує парадигму клієнт-сервер, тобто SQLite не є окремо працюючим процесом, з яким взаємодіє програма, а являє собою бібліотеку, з якої програма компонується, і SQLite стає складовою частиною програми. Таким чином, в якості протоколу обміну використовуються виклики функцій (API) бібліотеки SQLite. Такий підхід зменшує накладні витрати, час відгуку і спрощує програму. SQLite зберігає всю базу даних (включаючи визначення, таблиці, індекси і дані) в єдиному стандартному файлі на тому комп'ютері, на якому виповнюється програма.

Існує багато переваг використання SQLite як формату файлу програми, включаючи:

1. Кращі показники
    - Читання та запис із бази даних SQLite часто швидше, ніж читання та запис окремих файлів з диска

- Додаток має лише завантажувати потрібні йому дані, а не читати весь файл і проводити повний аналіз в пам'яті.
- Невеликі редагування перезаписують лише ті частини файлу, які змінюються, скорочуючи час запису та знос на SSD-накопичувачах.
- Зниження вартості та складності застосування
- Немає коду вводу / виводу файлу програми для запису та налагодження.
- Доступ до вмісту можна оновлювати за допомогою стислих запитів SQL замість тривалих та схильних до помилок процедурних процедур.
- Формат файлу може бути розширений у майбутніх випусках, просто додаючи нові таблиці та / або стовпчик, зберігаючи зворотну сумісність.
- Програми можуть використовувати повнотекстовий пошук та індекси RTREE та використовувати тригери для впровадження автоматизованого скасування / повторення.

2. Надійність
    - Вміст можна постійно оновлювати, так що при втраті електроенергії або збоїв в роботі не втрачається мало або нічого не відбувається.
    - Помилки набагато рідше в SQLite, ніж у власно написаному файлі коду вводу-виводу.
    - SQL-запитів у багато разів менше, ніж еквівалентний процедурний код, і оскільки кількість помилок на рядок коду є приблизно постійною, це означає менше помилок у цілому.
3. Доступність
    - Вміст бази даних SQLite можна переглядати за допомогою різноманітних сторонніх інструментів.

- Вміст, що зберігається в базі даних SQLite, швидше за все буде відновлений десятиліттями в майбутньому, задовго після того, як всі сліди оригінальної програми були втрачені.
- Файли баз даних SQLite рекомендовані Бібліотекою Конгресу США як формат зберігання для довготривалого збереження цифрового контенту.

3.3 PyCharm

PyCharm — інтегроване середовище розробки для мови програмування Python. Надає засоби для аналізу коду, графічний відладчик та інструмент для запуску юніт-тестів.

- Статичний аналіз коду, підсвічування синтаксису і помилок.
- Навігація по проекту і вихідного коду: відображення файлової структури проекту, швидкий перехід між файлами, класами, методами і використанням методів.
- Рефакторинг: перейменування, вилучення методу, введення змінної, введення константи, підйом і спуск методу і т.д.
- Вбудований відладчик для Python
- Вбудовані інструменти для юніт-тестування
- Розробка з використанням Google App Engine
- Підтримка систем контролю версій: загальний користувальницький інтерфейс для Mercurial, Git, Subversion, Perforce і CVS з підтримкою списків змін і злиття

3.4 QtDesigner

Qt Designer — вільне середовище для розробки графічних інтерфейсів (GUI) для програм, які використовують бібліотеку Qt.

Qt Designer дозволяє створювати графічні інтерфейси користувача за допомогою ряду інструментів. Існує панель інструментів «Панель віджетів», в якій доступні для використання елементи інтерфейсу — віджети, такі як, наприклад, ComboBox, «поле введення» LineEdit, «кнопка» PushButton і багато інших. Кожен віджет має свій набір

властивостей, що визначається відповідним йому класом бібліотеки Qt. Властивості віджета можуть бути змінені за допомогою «Редактора властивостей». Для кожного класу властивостей віджета існує свій спеціалізований редактор.

Qt Designer - інструмент для проектування і створення графічних користувацьких інтерфейсів (GUI) з компонентів Qt. Ви можете створити і налаштувати свої віджети або діалоги в режимі "що бачиш, то і отримаєш" (WYSIWYG).

Віджети та форми, створені за допомогою Qt Designer, безшовно інтегровані з керуючим кодом, який використовує механізм сигналів та слотів Qt, який дозволяє вам легко встановити поведінку до графічних елементів. Всі властивості встановлені в Qt Designer можна змінити динамічно всередині коду. Крім того, такі можливості як просування (promotion) віджетів і призначені для користувача модулі дозволяє вам використовувати з Qt Designer ваші власні компоненти.

4.Опис програмної реалізації

4.1 Концептуальна модель бази даних

На рисунку 4.1 зображена концептуальна модель бази даних[9], яка включає у себе наступні таблиці: користувачі, записи користувачів, файли користувачів, записи якими поділилися користувачі системи, та логування серверних подій.

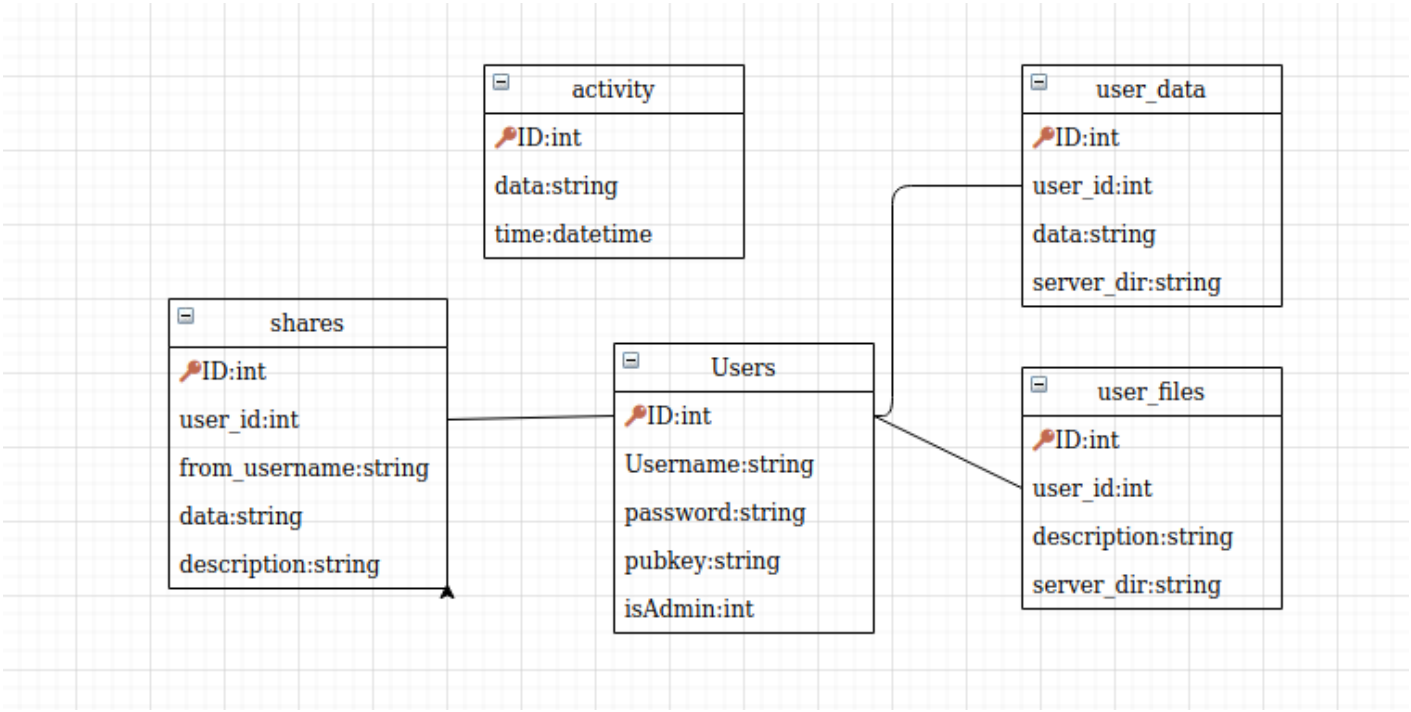

Рисунок 4.1 — Концептуальна модель БД.

4.2 Діаграми класів

4.2.1 Ядро клієнтського додатку

На рисунку 4.2.1.1 зображена діаграма класів для ядра клієнтського додатку[11].

Реалізовано клас для відправки абстрактних GET та POST запитів. Від нього успадковане кілька API контролерів для запитів пов’язаних із записами користувача, запитів пов’язаних із записами, якими поділилися, та адміністративного функціоналу.

Реалізовано класи для реалізації синхронної (AES) та асинхронної криптографії.

Створений PyQT клас для відображення JSON файлів у вигляді TreeView.

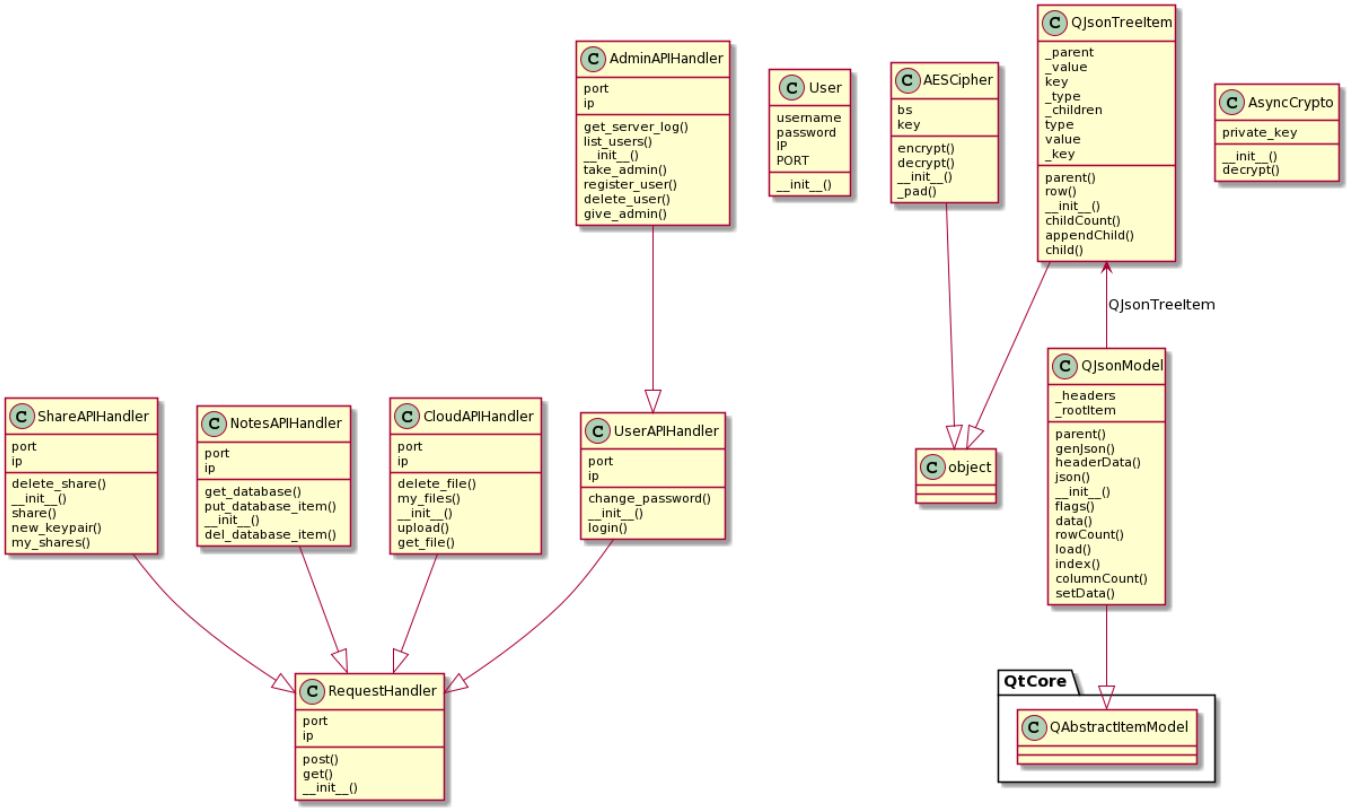

Рисунок 4.2.1.1 — Ядро клієнтської частини.

4.2.2 Ядро серверної частини

На рисунку 4.2.2.1 зображена діаграма класів для ядра серверного додатку.

Реалізовано абстрактний клас для роботи із базою даних, від нього успадковані класи для конкретних дій у БД — управління користувачами і різними типами записів.

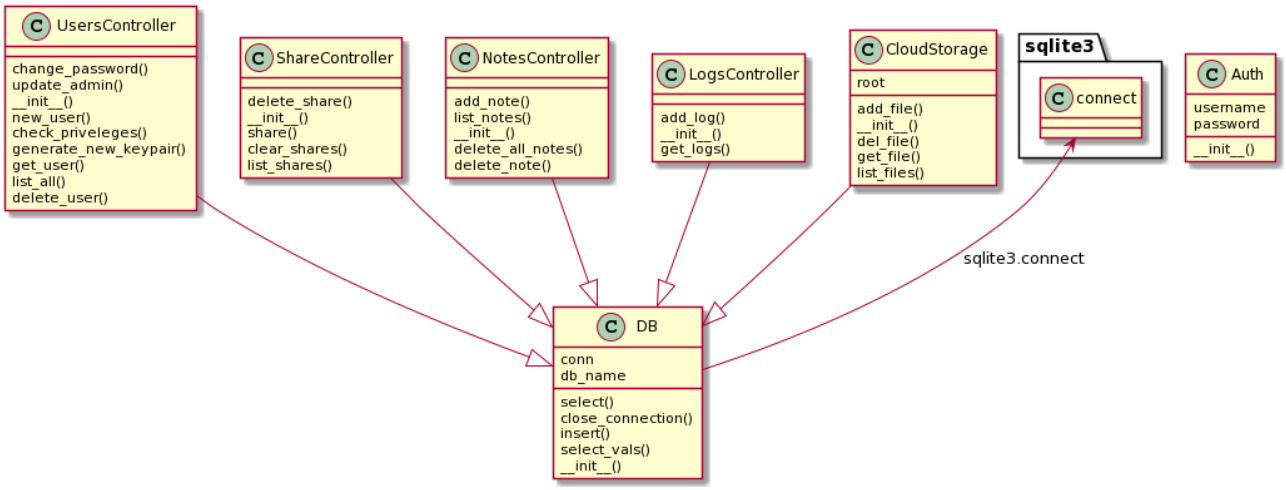

Рисунок 4.2.2.1 — Ядро серверної частини

4.2.3 Класи графічних форм

На рисунку 4.2.3.1 зображена діаграма класів які контролюють графічні форми (вікна) клієнтської програми. По одному класу на кожне вікно.

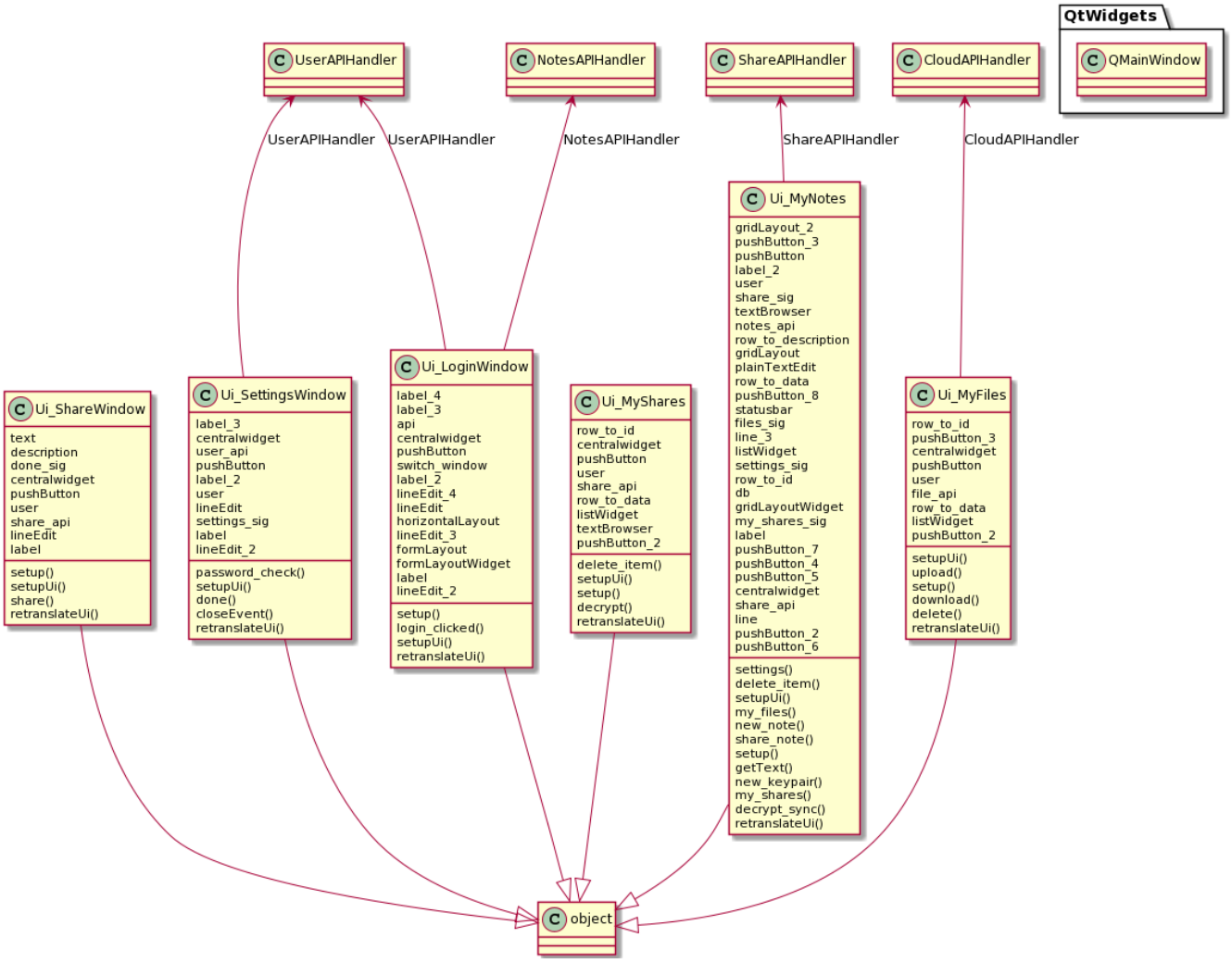

Рисунок 4.2.3.1 — Діаграма класів форм.

4.3 USE-CASE діаграма

На рисунку 4.3.1 зображена діаграма Use-Case[11].

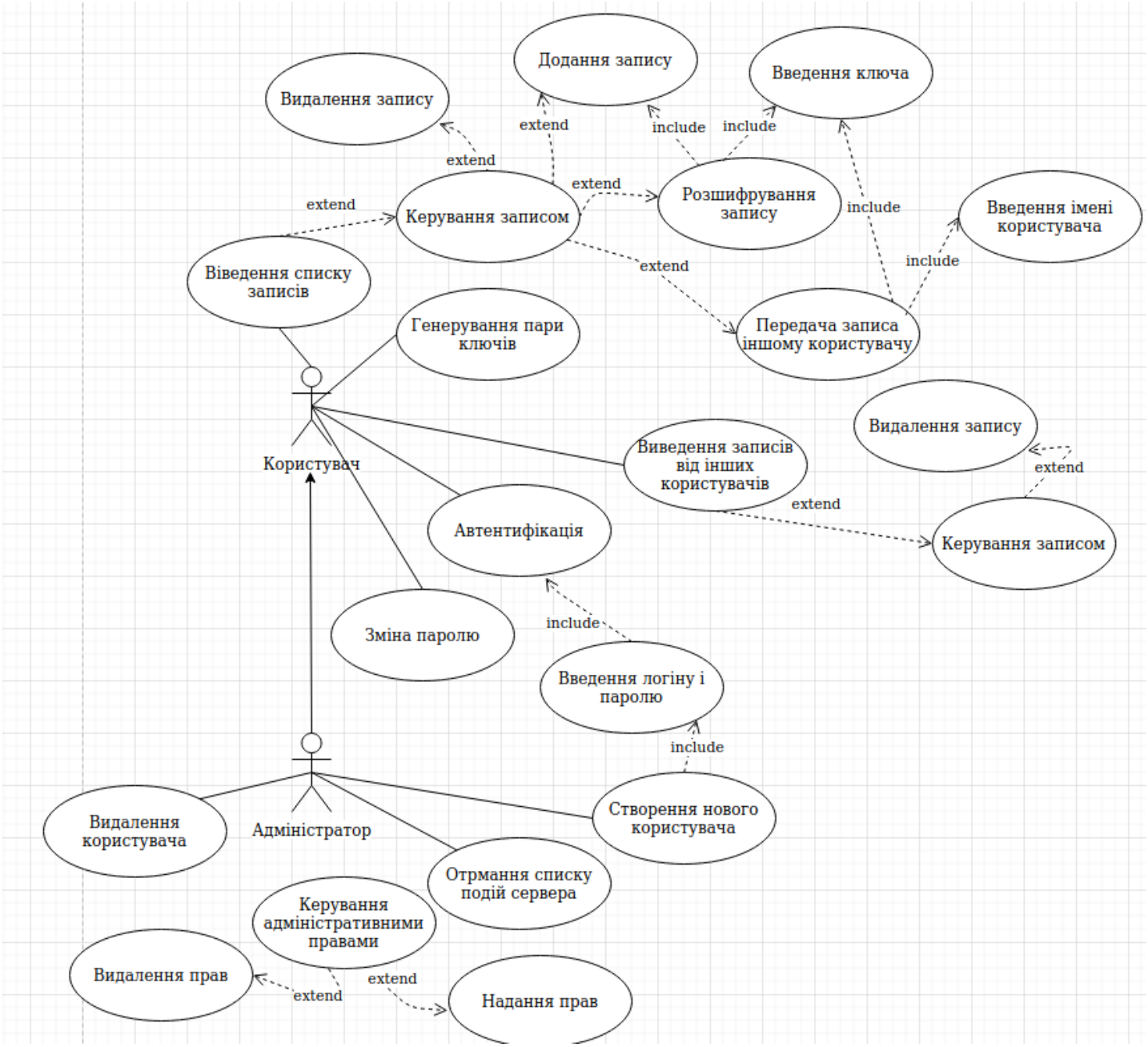

Рисунок 4.3.1.

4.4 Basic авторизація

4.4.1. Base64

Base64 - алгоритм кодування, який дозволяє перетворити будь-які символи в алфавіт, який складається з латинських літер, цифр, плюс і косою рисою. Завдяки ньому ви можете конвертувати китайські символи, смайли та навіть зображення в "читабельний" рядок, яку можна зберегти або перенести куди завгодно.

Щоб образно зрозуміти, чому був винайдений Base64, уявіть, що під час телефонного дзвінка Аліса хоче надіслати зображення Бобу. Перша проблема полягає в тому, що вона не може просто описати, як виглядає зображення, оскільки Боб потребує точної копії. У цьому випадку Аліса може перетворити зображення у двійкову систему і продиктувати Бобу двійкові цифри (біти), після чого він зможе перетворити їх назад у вихідне зображення. Друга проблема полягає в тому, що тарифи на телефонні дзвінки занадто дорогі і диктувати кожен байт буде довго. Щоб зменшити витрати, Аліса та Боб погоджуються використовувати більш ефективний метод передачі даних, використовуючи спеціальний алфавіт, який замінює кожні «шість цифр» однією літерою.

Щоб зрозуміти різницю, перегляньте зображення 5x5, перетворене у двійкові цифри:

010001 110100 100101 000110 001110 000011 011101 100001 000000 010000 000000 000001 000000 001111 000000 000000 000000 001111 111100 000000 000000 000000 000000 000000 000000 000010 110000 000000 000000 000000 000000 000000 000000 010000 000000 000001 000000 000000 000000 000000 000010 000000 1000000

Хоча те саме зображення, перетворене на Base64, виглядає приблизно так:

R0lGODdhAQABAPAAAP8AAAAACwAAAAAQABAAACAkQBAD

4.4.2 Авторизація

Схема “базової” HTTP-аутентифікації визначена в RFC 7617[12], який передає облікові дані у вигляді пар ідентифікаторів користувача / паролів, закодованих за допомогою base64. Для убезпечення від перехоплення даних для входу, які передаються, у програмному продукті використане примусове https з’єднання. На рисунку 4.4.2.1 зображений стандарт схеми авторизації типу «basic».

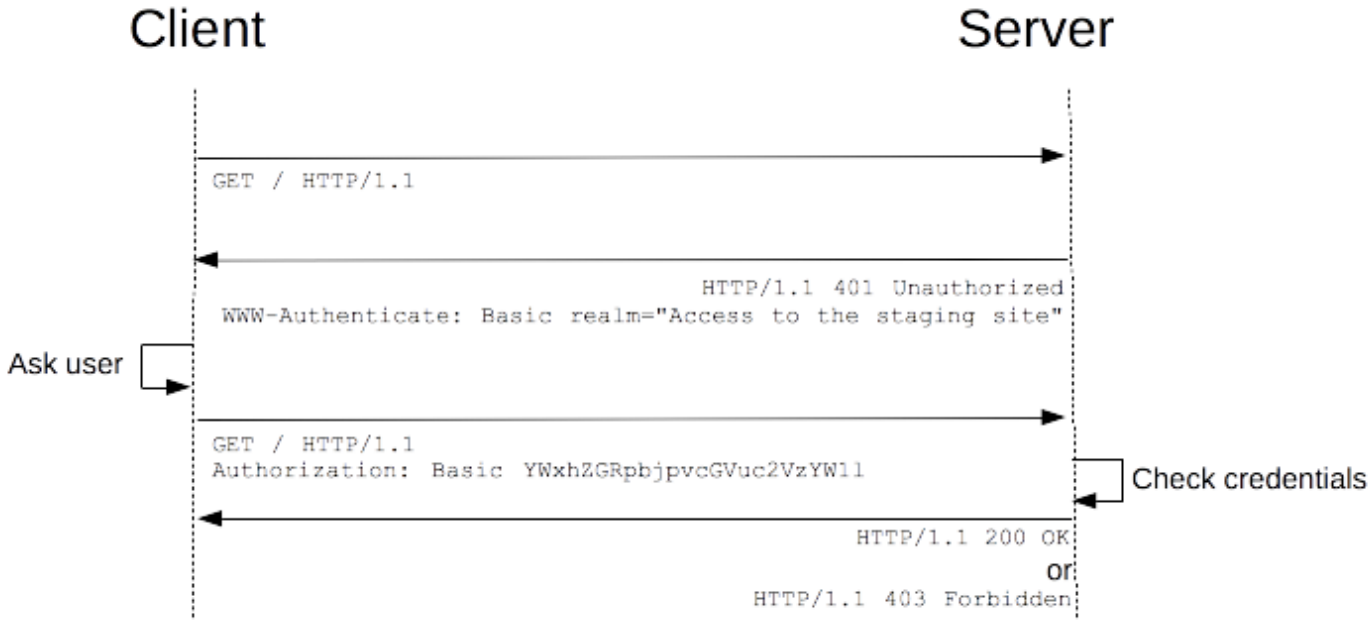

Рисунок 4.4.2.1

Для реалізації такої схеми авторизації був використаний вбудований функціонал веб-серверу Flask.

4.5 REST архітектура

4.5.1. Загальні відомості

REST (від англ. Representational State Transfer — «передача стану уявлення») — архітектурний стиль взаємодії компонентів розподіленого додатка в мережі. REST є узгоджений набір обмежень, що враховуються при проектуванні розподіленої гіпермедіа-системи[13].

Центральною абстракцією в REST є ресурс. А головні обмеження виглядають так:

- Клієнт-серверна модель

- Взаємодія без збереження стану
- Логічний інтерфейс

В архітектурі REST сервер не повинен зберігати ніякої інформації про стан операції. Сесії повинен зберігати клієнт. Це означає, що якщо сервер отримав два різних запити від одного клієнта, вони не повинні впливати один на одного. Через це, всю інформацію, потрібну для здійснення дії, клієнт повинен відправляти відразу.

Загальна схема роботи додатків із REST-API вказана на рисунку 4.5.1.1.

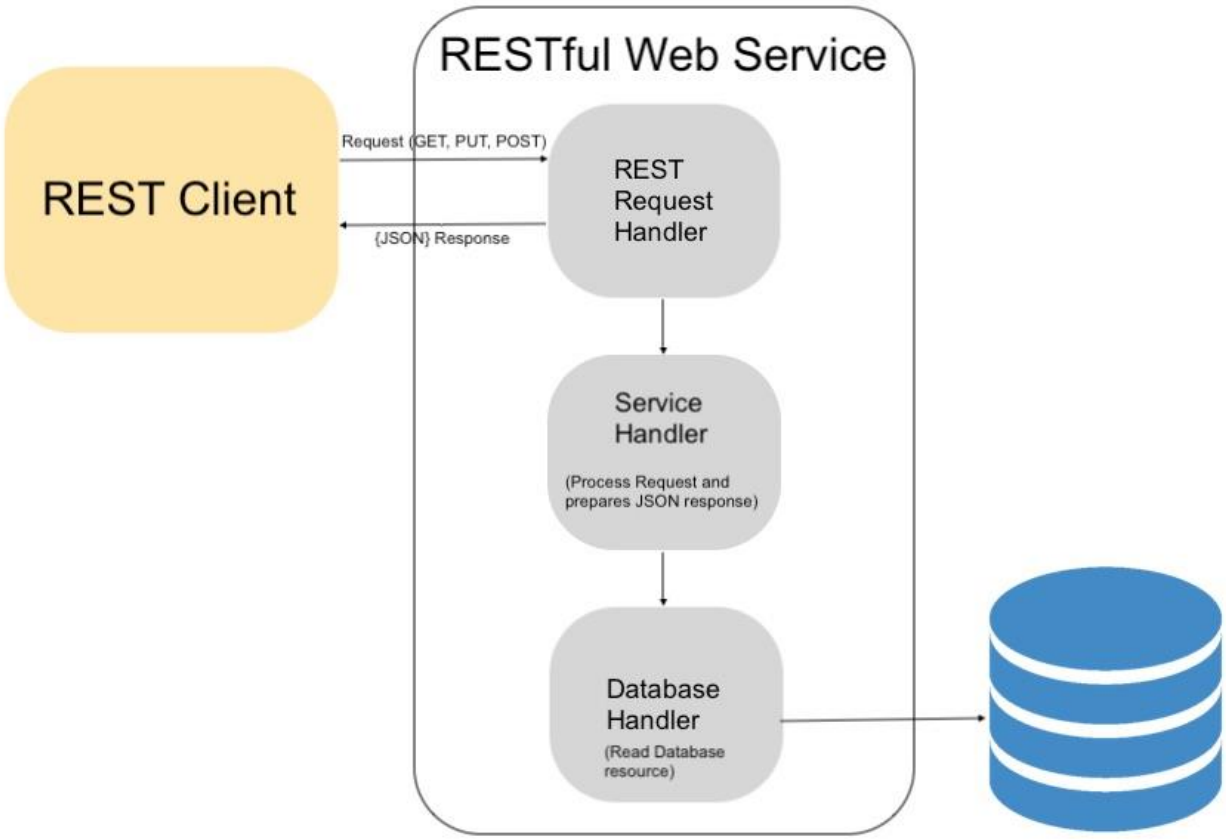

Рисунок 4.5.1.1

4.5.2 Реалізовані методи

- /login – basic аутентифікація
- /get_db – отримання записів поточного користувача
- /put_db_item – додати запис

- /del_db_item – видалити запис
- /change_pass – зміна паролю поточного користувача
- /register_user – реєстрація нового користувача
- /delete_user – видалити користувача
- /list_users – список зареєстрованих користувачів системи
- /give_admin – надати права адміністратора для користувача
- /take_admin – видалити права адміністратора у користувача
- /new_keypair генерування нової пари ключів для поточного користувача
- /share поділитися записом із користувачем
- /my_shares список і зміст (зашифрований) записів, якими поділилися із поточним користувачем
- /delete_share – видалити запис із списку записів, якими поділилися із поточним користувачем
- /clear_shares – очистити список записів, якими поділилися із поточним користувачем
- /log – лог записи серверу

4.6. Синхронна криптографія. Шифр AES.

Advanced Encryption Standard (AES), також відомий як Rijndael — симетричний алгоритм блочного шифрування (розмір блоку 128 біт, ключ 128/192/256 біт), прийнятий в якості стандарту шифрування урядом США за результатами конкурсу AES[14]. У червні 2003 року Агентство національної безпеки США постановив, що шифр AES є досить надійним, щоб використовувати його для захисту відомостей, що становлять державну таємницю (англ. Classified information).

У програмному продукті цей шифр використовується на боці клієнта, а сервер має лише зашифровані записи. Це забезпечує конфіденційність і захист від витоку інформації при компрометації серверу. Загальна схема роботи шифру вказана на рисунку 4.6.1.

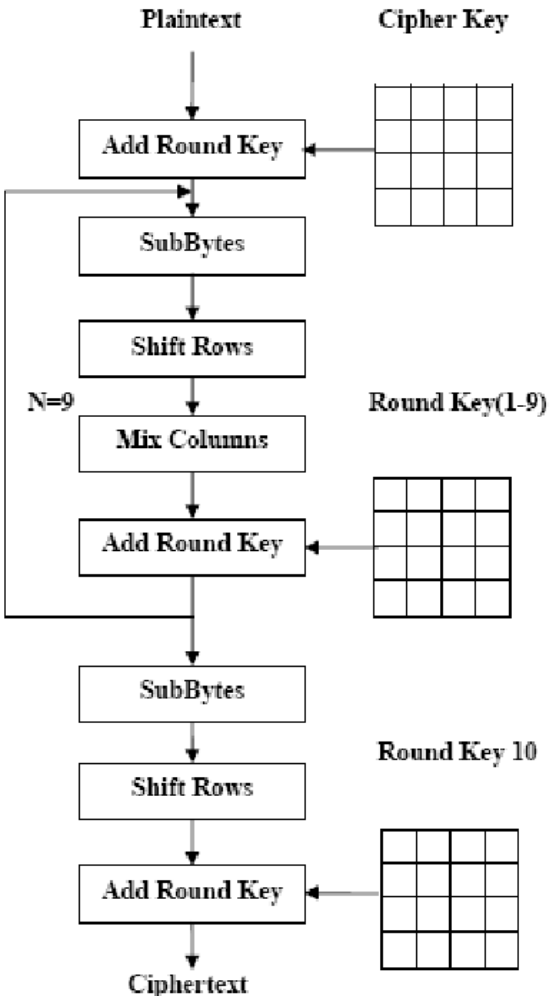

Рисунок 4.6.1

AES включає три блокові шифри: AES-128, AES-192 і AES-256.

AES-128 використовує 128-бітну довжину ключа для шифрування та дешифрування блоку повідомлень, тоді як AES-192 використовує 192-бітну довжину ключа, а AES-256 - 256-бітну довжину ключа для шифрування та дешифрування повідомлень. Кожен шифр шифрує та розшифровує дані в блоках по 128 біт, використовуючи криптографічні ключі 128, 192 та 256 біт відповідно.

Симетричні шифри використовують один і той же ключ для шифрування та розшифрування, тому відправник і одержувач повинні знати і використовувати один і той же секретний ключ.

Існує 10 раундів для 128-бітних ключів, 12 раундів для 192-бітних ключів і 14 раундів для 256-бітних ключів. Раунд складається з декількох етапів обробки, які включають підстановку, транспозицію та змішування вхідного простого тексту для перетворення його на кінцевий вихід шифротексту.

NIST вказав, що новий алгоритм AES повинен бути шифром блоку, здатним обробляти 128-бітні блоки, використовуючи клавіші розміром 128, 192 та 256 біт. Інші критерії вибору як наступного алгоритму AES включали наступне:

- Безпека. Алгоритми змагань повинні були судити про їх здатність протистояти атаці - порівняно з іншими представленими шифрами. Сила безпеки повинна була вважатися найважливішим фактором конкуренції.
- Вартість. Заплановані для випуску на глобальній, неексклюзивній та безоплатній основі, алгоритми кандидатів повинні були оцінюватися на основі обчислювальної та оперативної пам'яті.
- Впровадження. Фактори, які слід враховувати, включали гнучкість алгоритму, придатність для апаратного чи програмного забезпечення та загальну простоту.

4.7. Асинхронна криптографія

Для реалізації безпечного функціоналу, за допомогою якого користувачі системи зможуть ділитися своїми записами була обрана асинхронна криптографія, а саме – криптографічна система з відкритим ключем.

Криптографічна система з відкритим ключем — система шифрування і / або електронного підпису (ЕП), при якій відкритий ключ передається по відкритому (тобто незахищені, доступному для спостереження) каналу і використовується для перевірки ЕП і для шифрування повідомлення. Для генерації ЕП і для розшифровки повідомлення використовується закритий ключ[15]. Асиметричне шифрування з відкритим ключем базується на наступних принципах:

- Можна згенерувати пару дуже великих чисел (відкритий ключ і закритий ключ) так, щоб, знаючи відкритий ключ, не можна було вирахувати закритий ключ за розумний термін. При цьому механізм генерації є загальновідомим.
- Є надійні методи шифрування, що дозволяють зашифрувати повідомлення відкритим ключем так, щоб розшифрувати його можна було тільки закритим ключем. Механізм шифрування є загальновідомим.
- Власник двох ключів нікому не повідомляє закритий ключ, але передає відкритий ключ контрагентам або робить його загальновідомим.

Якщо необхідно передати зашифроване повідомлення власнику ключів, то відправник повинен отримати відкритий ключ. Відправник шифрує своє повідомлення відкритим ключем одержувача і передає його одержувачу (власнику ключів) по відкритих каналах. При цьому розшифрувати повідомлення не може ніхто, крім власника закритого ключа. Загальна високорівнева схема роботи цього шифру вказана на рисунку 4.7.1.

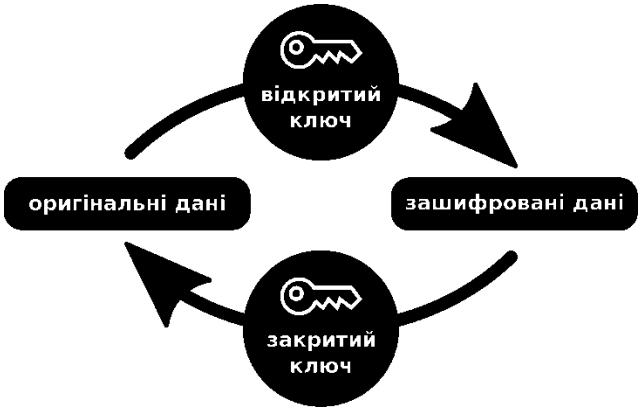

Рисунок 4.7.1 — Криптографія з відкритим ключем.

4.8. JSON

За рахунок своєї лаконічності в порівнянні з XML формат JSON може бути використаній для серіалізації складних структур, наприклад об’єктів класу[16]. Застосовується в веб-додатках як для обміну даними між браузером і сервером (AJAX), так і між серверами (програмні HTTP-сполучення).

JSON-текст представляє собою набір пар ключ:значення.

Саме такий тип представлення був вибраній для обміну інформацією між сервером та клієнтом. Це забезпечує гнучкість системи і дозволяє легко написати клієнти для інших платформ, тому що більшість мов програмування підтримує дереалізацію JSON.

4.9. Безпечне зберігання даних для входу. Хешування паролів

4.9.1 Загальні відомості

Зберігання паролів користувачів є важливою складовою для будь-якої веб-програми. Під час зберігання пароля користувача ви повинні переконатися, що він захищений таким чином, що якщо ваші дані порушені, ви не втратите пароль свого користувача.

Було багато відомих випадків веб-сайтів та веб-додатків, які піддавали компрометації їхні дані користувачів. Це стає ще гірше, коли розробники веб-сайту не зберегли пароль користувача в безпечний спосіб. Під час розробки веб-сайтів чи веб-

додатків важливо мати завжди точку зору, що всі бажають зламати ваш сервер. Як правило, ви повинні вважати, що кожна точка вашого коду вразлива для атаки. Ви можете подумати, "немає жодного способу отримати доступ до моєї бази даних!", Але це не правильне ставлення.

Зберігаючи пароль у базі даних, він не повинен бути у простому тексті. Збереження пароля в простому тексті означитиме, що кожен, хто переглянув базу даних, зможе просто прочитати паролі користувача.

4.9.2 Хеш-алгоритми

Хеш-алгоритми є односторонніми функціями. Вони перетворюють будь-яку кількість даних в «дактилоскопічний відбиток» фіксованої довжини, який не може бути звернемо[15].

Загальний підхід до безпечного зберігання паролів - "хешування" паролів. "Хеш" - це одностороння функція, яка генерує подання пароля. Отже, коли користувач підписується на обліковий запис і обирає пароль, він зберігається як генерований хеш, а не фактичні символи, які ввів користувач. Коли ви запускаєте пароль через функцію хешування, він завжди створюватиме той же вихід. Коли користувач намагається увійти за допомогою своєї електронної пошти та пароля, введений пароль повторно хешується, а потім порівнюється з тим, що зберігається в базі даних. Якщо два хеші однакові, користувач ввів правильний пароль.

Хеші неможливо перетворити назад у звичайний текст, але вам не потрібно їх перетворювати назад, щоб зламати їх. Коли ви знаєте, що певна рядок перетворюється на певний хеш, ви знаєте, що будь-який екземпляр цього хеша являє собою цей рядок.

Це відмінно підходить для захисту паролів, тому що ми хочемо зберегти паролі в зашифрованому вигляді, неможливе для розшифровки, і в той же час ми повинні бути здатні перевірити те, що пароль користувача коректний.

1. Користувач створює обліковий запис;
2. Обчислюється хеш-код від пароля і зберігається в базі даних.

3. Коли користувач намагається увійти в систему, хеш-код, обчислений від пароля, який він ввів, порівнюється з хеш-кодом реального пароля (витягнутим з бази даних);
4. Якщо хеш-коди збігаються, користувачеві надається доступ. Якщо немає — користувачеві повідомляється, що він ввів невірні дані для входу в обліковий запис;
5. Кроки 3 і 4 повторюються кожен раз, коли хтось намагається увійти в свій аккаунт.

Для того щоб ускладнити викриття хешу, також потрібно посолити його. Засолення - це те, коли додається довільні дані до пароля, перш ніж він хешується.

Коли користувач повертається до вашої системи, ви просто береться його введений пароль, додається сіль і потім хеш, щоб побачити, чи відповідає він хешу, який був збережений.

4.9.3 PBKDF2

У програмному продукті використовується функція хешування PBKDF2.

PBKDF2 (англ. Password-Based Key Derivation Function) - стандарт формування ключа на основі пароля. Є частиною PKCS # 5 v2.0 (RFC 2898). Замінив PBKDF1, який обмежував довжину породжується ключа 160 бітами.

PBKDF2 використовує псевдовипадкову функцію для отримання ключів. Довжина ключа не обмежується (хоча ефективна потужність простору ключів може бути обмежена особливостями застосовуваної псевдовипадковою функції). Використання PBKDF2 рекомендовано для нових програм і продуктів. Як псевдовипадкова може бути обрана криптографічна хеш-функція, шифр, HMAC.

Коли стандарт був написаний у 2000 році, рекомендована мінімальна кількість ітерацій становила 1 000, але цей параметр передбачається збільшувати з часом із збільшенням швидкості процесора. Стандарт Kerberos в 2005 році рекомендував 4 096 повторень; Apple, як повідомляється, використовувало 2 000 для iOS 3 та 10 000 для iOS

4; тоді як LastPass у 2011 році використовував 5000 ітерацій для клієнтів JavaScript та 100 000 ітерацій для хешування на стороні сервера.

Додавання солі до пароля знижує можливість використовувати попередньо обчислені хеші для атак, і означає, що кілька паролів потрібно перевіряти індивідуально, не всі відразу. Стандарт рекомендує довжину солі не менше 64 біт. Національний інститут стандартів і технологій США рекомендує довжину солі в 128 біт.

4.10. HTTP методи для взаємодії з системою

Для взаємодії із системою по протоколу http(s) були використані найпоширеніші методи – GET та POST запити.

POST — один з багатьох методів запиту, підтримуваних HTTP протоколом, використовуваним у Всесвітній павутині. Метод запиту POST призначений для запиту, при якому веб-сервер приймає дані, ув'язнені в тіло повідомлення, для зберігання.

На рисунку 4.10.1 зображений вигляд і опис змісту запита цього типу.

Рисунок 4.10.1

На відміну від нього, метод HTTP GET призначений для отримання інформації від сервера. На рисунку 4.10.2 зображений вигляд запиту POST.

Рисунок 4.10.2

4.11.HTTPS

HTTPS (Hypertext Transfer Protocol Secure) - захищена версія протоколу HTTP, яка використовує протокол SSL / TLS для шифрування та автентифікації. HTTPS визначений RFC 2818 (травень 2000 р.) І використовує порт 443 за замовчуванням замість порту HTTP 80.

URL-адреса HTTPS починається з https: // замість http: //

Сучасні веб-браузери також вказують, що користувач відвідує захищений веб-сайт HTTPS, показуючи символ закритого замка зліва від URL-адреси (рисунок 4.11.1).

Рисунок 4.11.1

Оскільки HTTP спочатку був розроблений як чіткий текстовий протокол, він вразливий для підслуховування та атаки MITM (man-in-the-middle). HTTPS пом'якшує ці вразливості шляхом підсилення протоколу HTTP поверх SSL / TLS, так що всі повідомлення шифруються в обох напрямках між двома мережевими комп'ютерами (наприклад, клієнтом та веб-сервером). Хоча підслуховувач все ще може отримати доступ до IP-адрес, номерів портів, доменних імен, кількості обмінюваної інформації та тривалості сеансу, всі фактично обмінювані дані надійно зашифровані SSL / TLS, включаючи:

1. URL-адреса запиту (яку веб-сторінку запитував клієнт)
2. Вміст веб-сайту
3. Параметри запиту
4. Заголовки
5. Куки-файли

Через це HTTPS безпечно використовувати для передачі конфіденційної інформації, такої як номери кредитних карт, банківська інформація та номери соціального страхування через незахищені мережі, такі як Інтернет.

5. Робота користувача з програмною системою

5.1 Системні вимоги

Для роботи програми треба мати встановлений Python3 з наступними бібліотеками:

requests==2.18.4

cryptography==2.1.4

Werkzeug==1.0.0

Flask_HTTPAuth==3.3.0

Flask==1.1.1

pycryptodome==3.9.7

PyQt5==5.14.1

Це можна зробити скопіювавши текст зверху у текстовій файл requirements.txt та написав у командному рядку: *pip3 install -r requirements.txt.*

Приклад результату виводу команди зображений на рисунку 5.1.1.

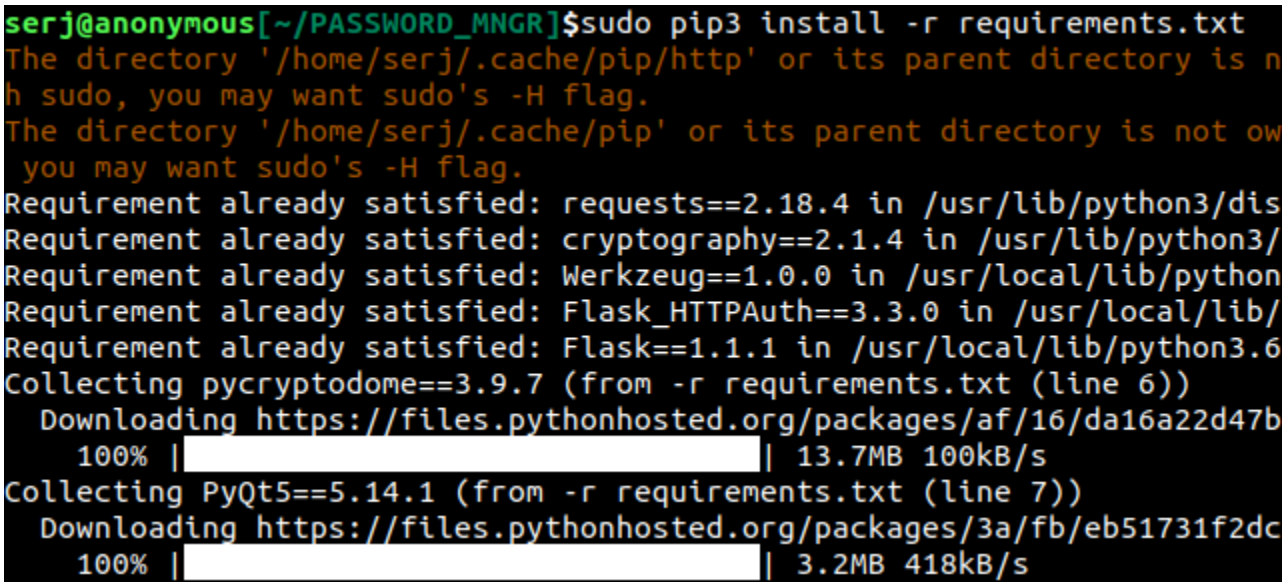

Рисунок 5.1.1

5.2 Запуск сервера

Команда для запуску сервера вказана на рисунку 5.2.1

```
serj@anonymous[~/PASSWORD_MNGR/Server]$python3 server.py
 * Serving Flask app "server" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on https://0.0.0.0:5000/ (Press CTRL+C to quit)
```

Рисунок 5.2.1

Після запуску можна перевірити чи правильно запустився сервер, наприклад визначивши процес, який слухає порт 5000. В операційній системі Linux це можна зробити командою *ss -tulpn | grep 5000* (рисунок 5.2.2).

```
serj@anonymous[~/PASSWORD_MNGR]$sudo ss -tulpn | grep 5000
tcp    LISTEN   0        128              0.0.0.0:5000          0.0.0.0:*      users:(("python3",pid=8569,fd=3))
```

Рисунок 5.2.2

5.3. Запуск клієнта

Команда для запуску клієнта: *python3 client.py*

Після коректного запуску можна побачити форму входу як на рисунку 5.3.1.

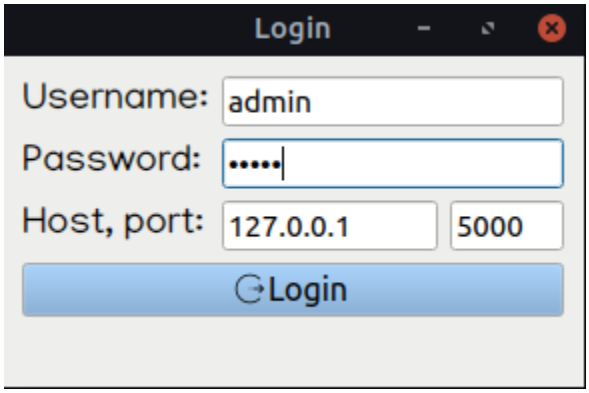

Рисунок 5.3.1

5.4. Процес роботи із програмою

5.4.1 Вхід

Якщо дані (логін, пароль, адреса сервера) введені вірно, тобто автентифікація пройдена успішно, з’явиться головне вікно (рисунок 5.4.1.1), яке включає в себе список записів, кнопки для видалення, розшифрування, додання записів. Також є елементи

управління, які дозволяють поділитися записом, відкрити вікно із записами, якими поділилися із поточним користувачем, згенерувати нову пару ключів та відкрити вікно налаштувань. Знизу присутній напис, якій повідомляє ім’я поточного користувача.

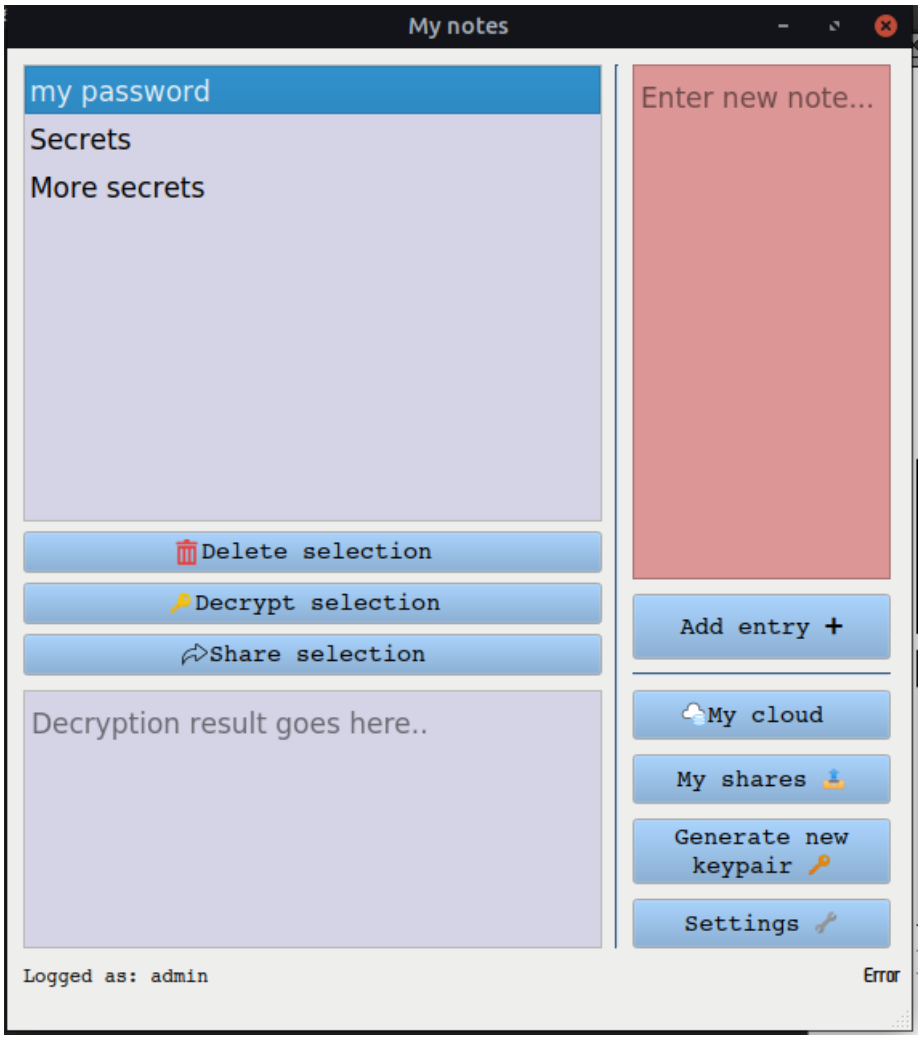

Рисунок 5.4.1.1

5.4.2 Додання запису

При натисканні кнопки «Add entry», якщо в полі “Enter new note…” э будь-який текст буде ініційований процес створення запису (рисунок 5.4.2.1).

Рисунок 5.4.2.1

Після вводу та підтвердження (рисунок 5.4.2.2) майстер ключу, буде запропоновано ввести опис до запису (рисунок 5.4.2.3).

Рисунок 5.4.2.2 – Підтвердження майстер ключу

Рисунок 5.4.2.3 – Введення опису

5.4.3. Розшифрування своїх записів

При натисканні на кнопку «Decrypt selection» починається процес розшифрування вибраного запису із списку (рисунок 5.4.3.1). З’являється вікно для вводу ключа, а при успішному розшифруванні результат з’являється у лівому нижньому квадраті (рисунок 5.4.3.2).

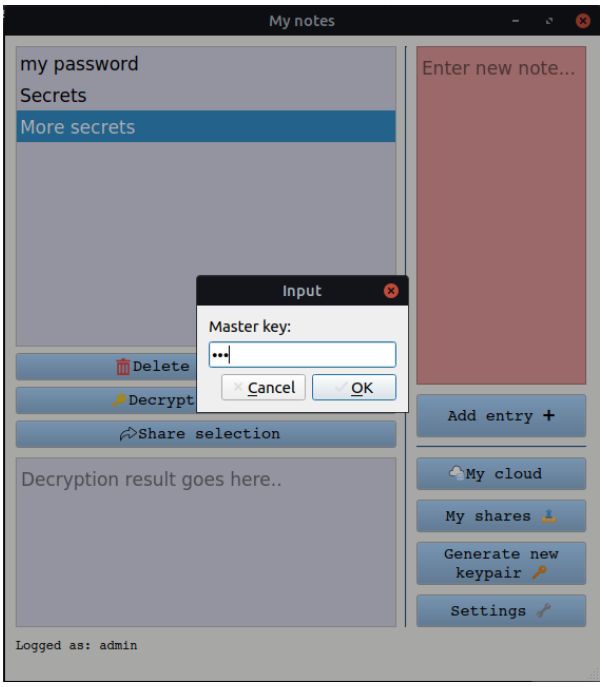

Рисунок 5.4.3.1

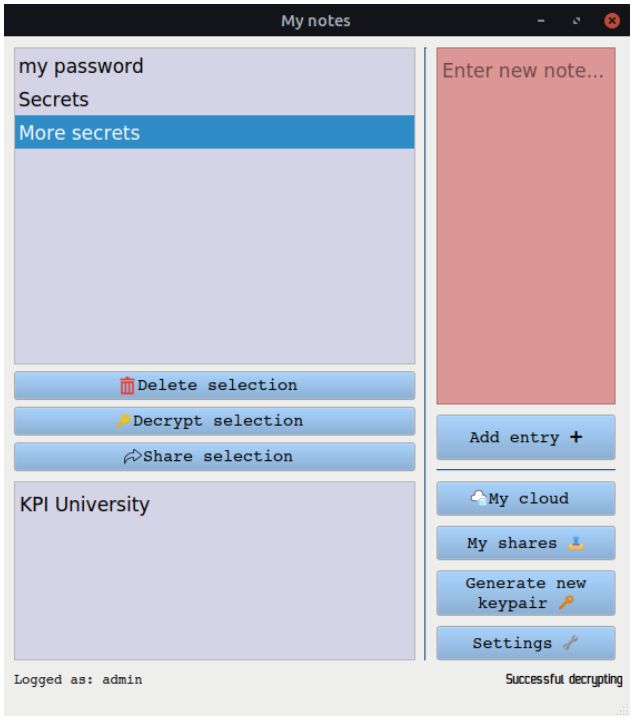

Рисунок 5.4.3.2

5.4.4. Демонстрація процесу безпечної передачі даних між користувачами

Для демонстрації процесу передачі записів між користувачами, авторизуємось як користувач “test”, згенеруємо нову пару ключів (натиснувши кнопку «Generate new keypair») та перевіримо, що список записів, якими поділилися із поточним користувачем – пустий, натиснувши кнопку «My shares». (Рисунок 5.4.4.1)

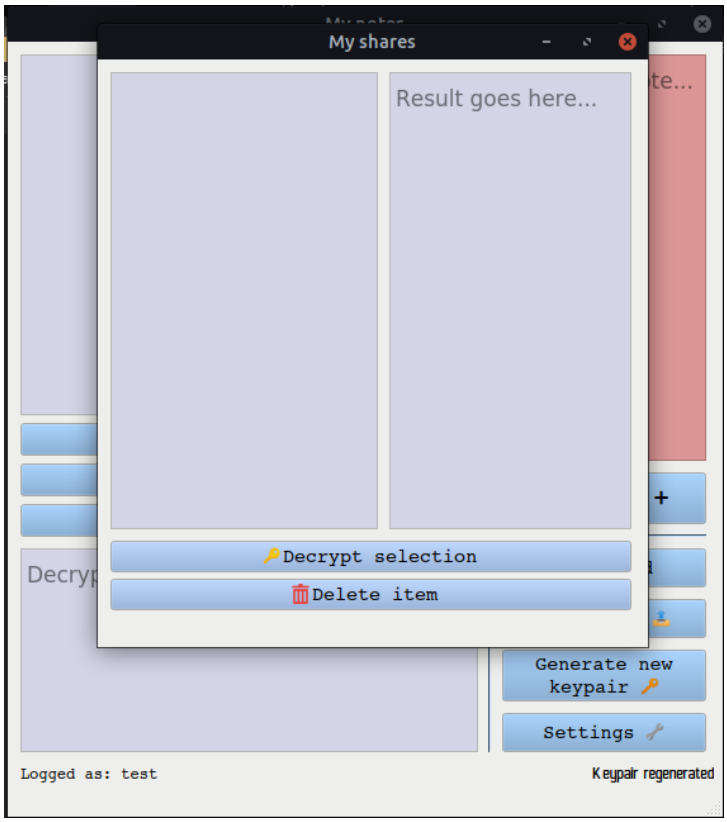

Рисунок 5.4.4.1

Далі потрібно зайти від імені іншого користувача, наприклад, «admin» та поділитися будь яким записом, натиснувши на кнопку «Share selection». Треба буде ввести ключ, та при умові коректного розшифрування з’явиться вікно для вводу імені користувача, якому призначений запис. (Рисунок 5.4.4.2)

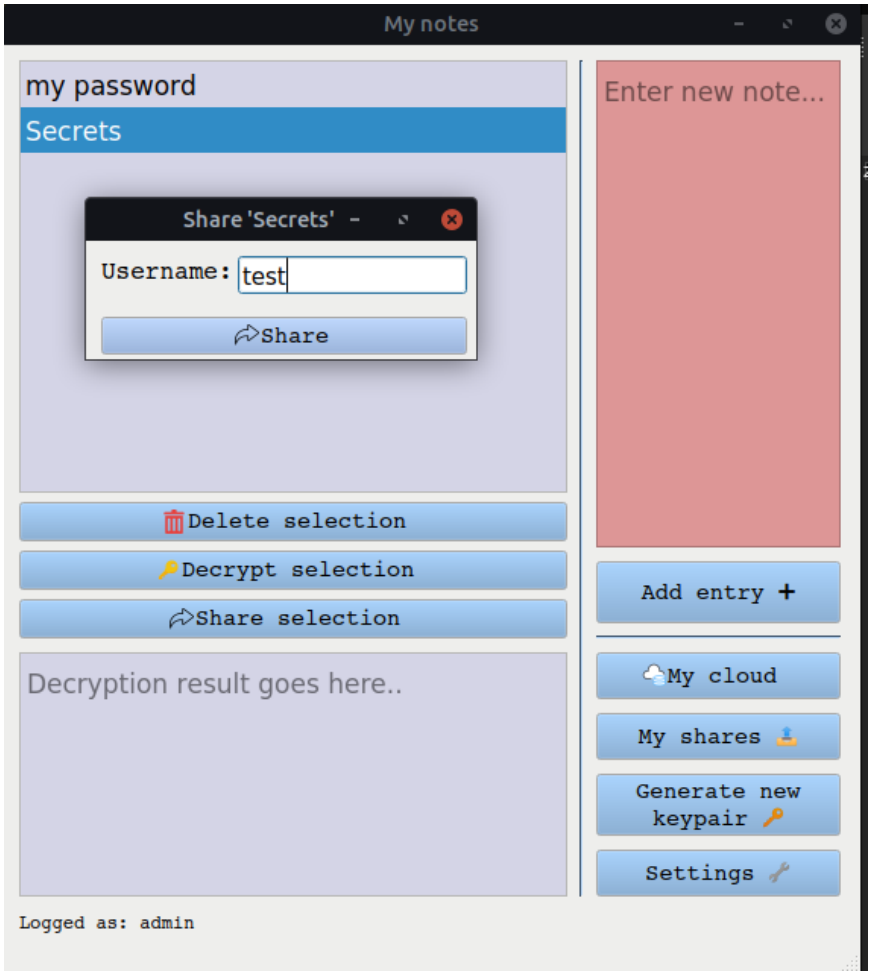

Рисунок 5.4.4.2

Після цього, авторизувавшись як “test” можна буде побачити новий запис у списку та розшифрувати його, натиснувши кнопку «decrypt selection». (Рисунки 5.4.4.3 та 5.4.4.4). Розшифрування відбувається автоматично, приватний ключ зберігається у директорії програми.

Рисунок 5.4.4.3

Рисунок 5.4.4.4

5.4.4. Зміна паролю

Кнопка налаштувань (Settings) відкриває вікно для зміни паролю – рисунок 5.4.4.1.

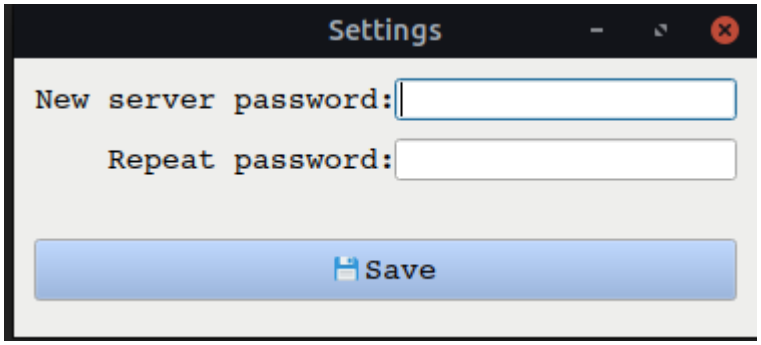

Рисунок 5.4.4.1

5.4.5 Адміністративний функціонал

Якщо у користувача є права адміністратора системи, то він зможе виконати автентифікацію у додатковому клієнті – *admin.py*

За умови вдалої автентифікації та авторизації можна побачити вікно керування серверною частиною (Рисунок 5.4.6.1). Воно включає в себе логи серверу, функціонал додавання користувача, видалення користувача та керування адміністративними правами.

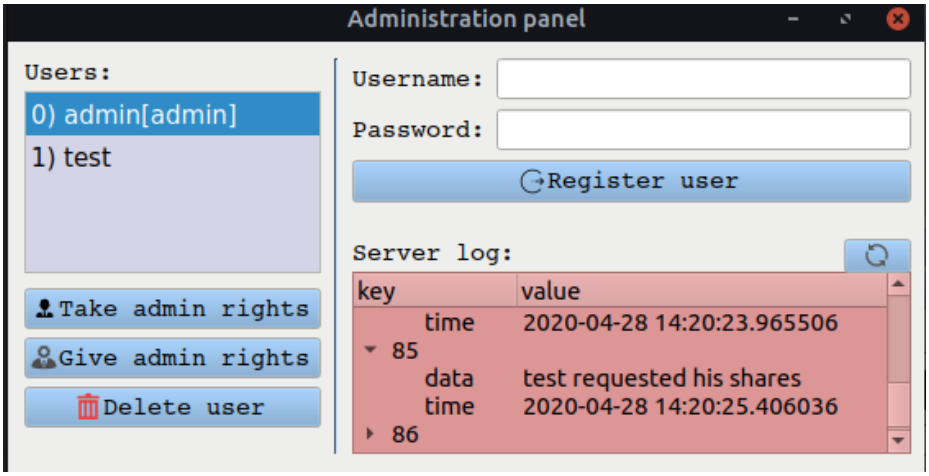

Рисунок 5.4.5.1

Процес додання користувача та надання йому адміністративних прав за допомогою кнопки «give admin rights», можна побачити на рисунках 5.4.5.2 та 5.4.5.3.

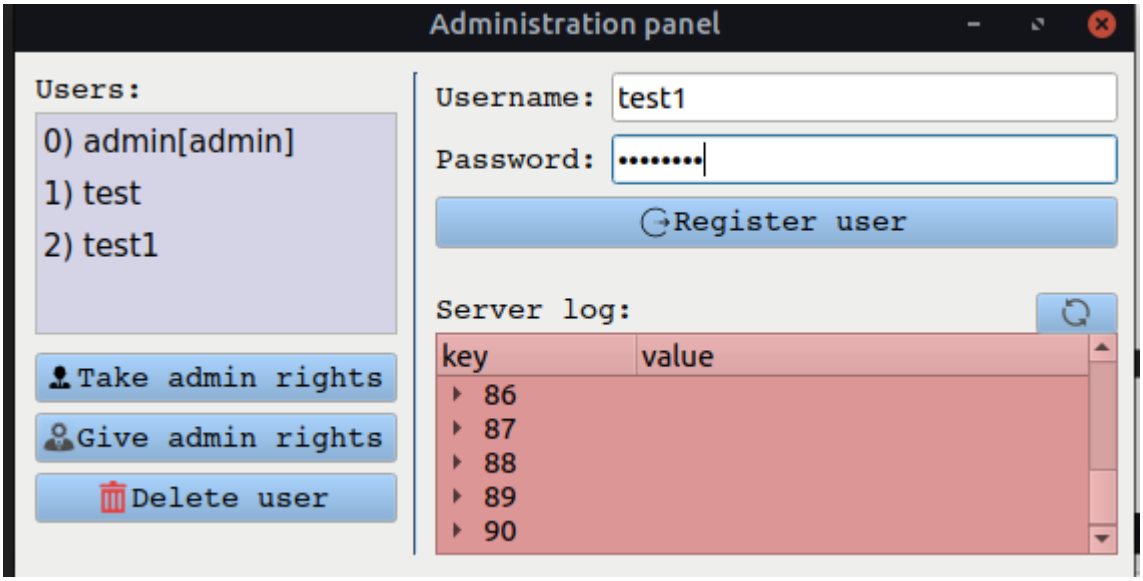

Рисунок 5.4.5.2 - Реєстрування нового користувача

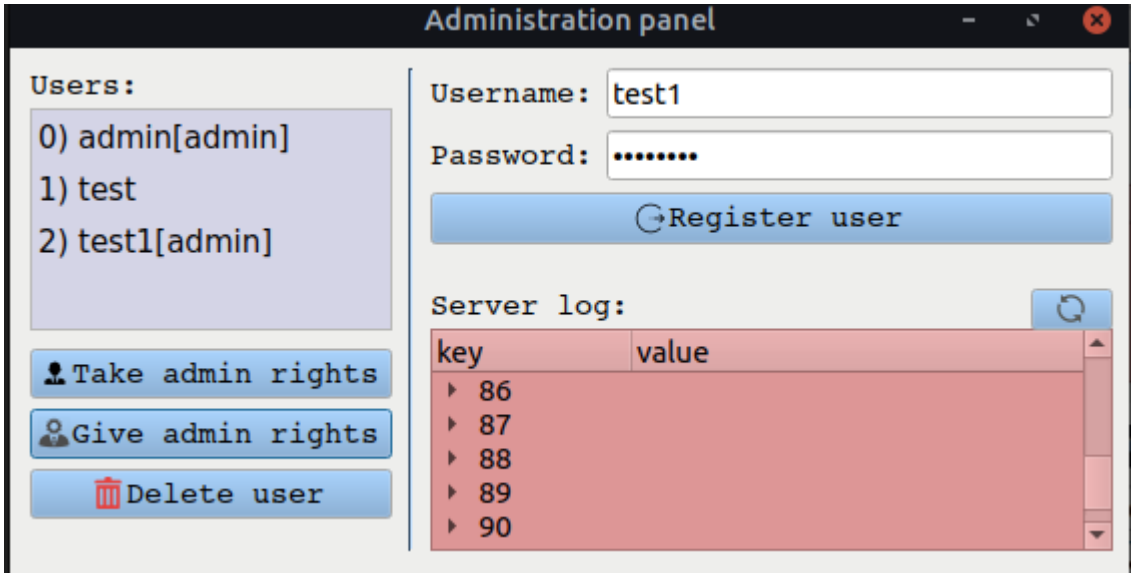

Рисунок 5.4.5.3 - Надання адміністративних прав користувачу «test1»

5.4.6. Додатковий функціонал – хмарне сховище.

Для підвищення функціональності програми був розроблений додатковий функціонал у вигляді хмарного сховища.

Будь-який зареєстрований користувач може завантажити файл у розділі «MyCloud» (рисунок 5.4.6.1 та 5.4.6.2) а потім з будь якого іншого пристрою скачати його, увійшовши у свій аккаунт (рисунок 5.4.6.3 та 5.4.6.4).

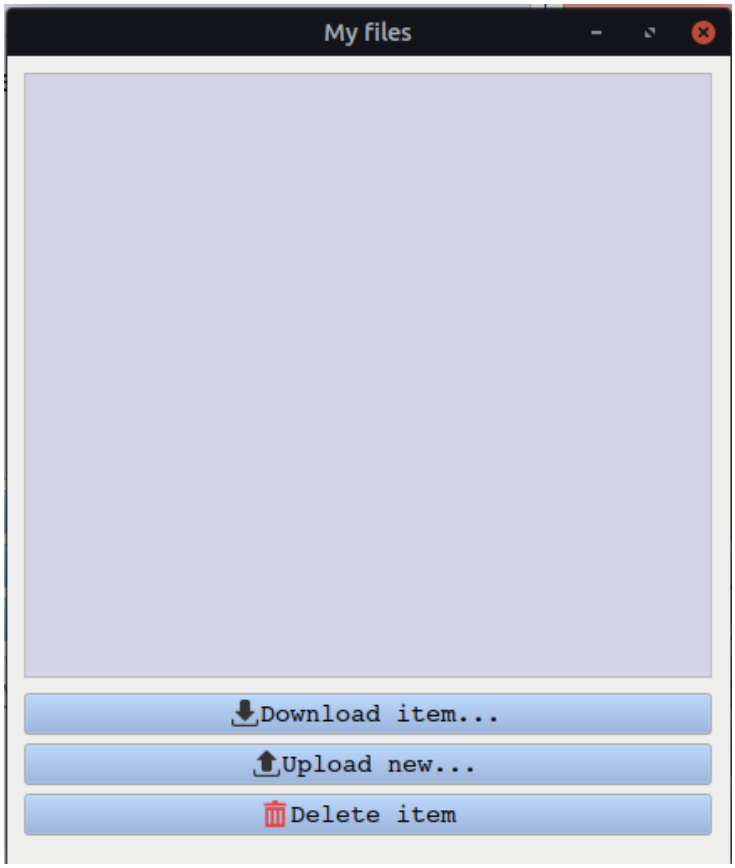

Рисунок 5.4.6.1

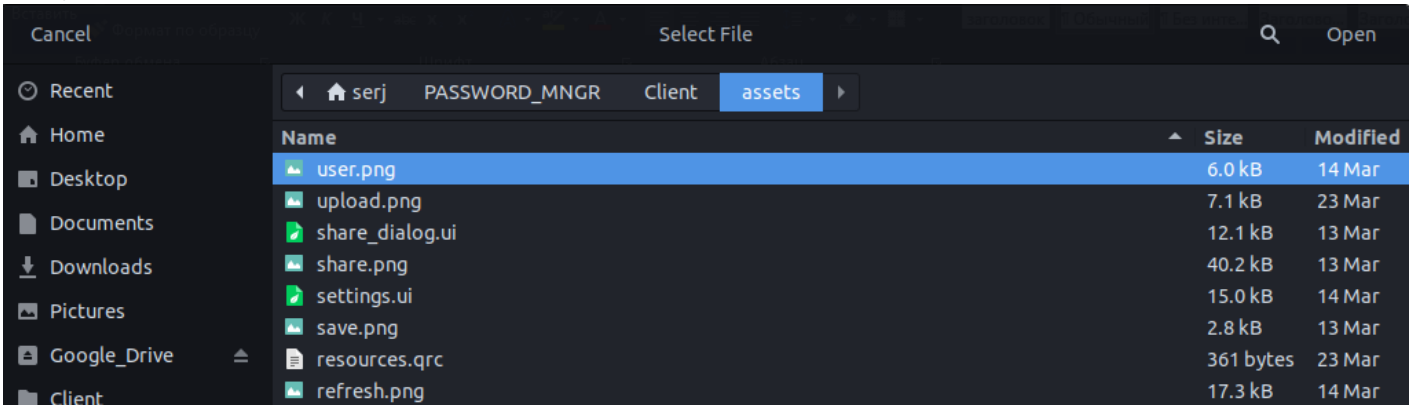

Рисунок 5.4.6.2 – Вибір файлу

Рисунок 5.4.6.3

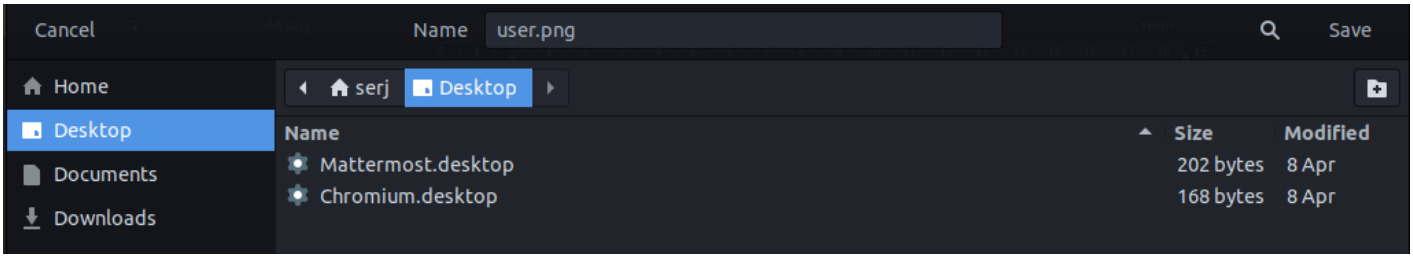

Рисунок 5.4.6.4 – Вибір директорії для збереження файлу

Висновки

Під час проектування був виконаний аналіз рішень на ринку, їх сильні та слабкі сторони. Маже повна відсутність автоматизованої системи яку можна назвати як «командний менеджер паролів» змусила розробити власну клієнт-серверну систему.

Для реалізації клієнт-серверного менеджера паролів із функціоналом безпечного надання доступу до даних входу іншим користувачам, була вибрана мова програмування Python. Серверна частина реалізована за допомогою сервера Flask. В якості СКБД при вирішенні даної задачі використовується SQLite. Клієнтська частина використовує відомий графічний фреймворк PyQt5.

Сервер реалізований у вигляді REST API, та забезпечує функціонал синхронізації із базою даних, отримання публічного ключа вибраного користувача та адміністративний інтерфейс. Це дозволяє імплементувати функціонал програми для будь-якої платформи. Сервер виконаний таким чином, що не зберігає приватних ключів користувачів, та має лише опис записів, без їхнього змісту. Завдяки такому підходу навіть при компрометації серверного програмного забезпечення, або бази даних, атакуючій не отримує чутливої інформації. Завдяки https з'єднанню майже неможливе виконання MITM атак.

В процесі розробки і проектування буди отримані корисні навички із криптографії, програмування та проектування.

Було реалізовано усі поставлені задачі, а саме:

- Безпечне зберігання даних, зашифрованих стійким алгоритмом, користувачу доступна синхронізація своїх записів із хмарним сховищем.
- Серверне ПЗ не має ключів для розшифровування даних користувачів, а лише публічну інформацію.
- Система дозволяє поділитися будь-яким своїм записом за допомогою асинхронної криптографії, кожному користувачу системи доступний функціонал генерації пар публічних та приватних ключів.

- Виділений сервер, який можна розгорнути навіть у локальній корпоративній мережі, тим самим захистивши його від атак зовні.

Розроблену програму можна використовувати для надання корпоративного доступу до чутливої інформації, наприклад між філіалами або при віддаленій роботі. Також програма може бути корисна експертам із кібербезпеки, які при тестуванні отримують дані для входу, але поважають конфіденційність клієнта.

Список використаних джерел

1. “Менеджер паролів” [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/%D0%9C%D0%B5%D0%BD%D0%B5%D0%B4%D0%B6%D0%B5%D1%80_%D0%BF%D0%B0%D1%80%D0%BE%D0%BB%D1%96%D0%B2
2. Слабые места современных менеджеров паролей [Електронний ресурс] – Режим доступа:https://xakep.ru/2014/09/08/password-manager-pentest/
3. “Злом пароля” [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/%D0%97%D0%BB%D0%BE%D0%BC_%D0%BF%D0%B0%D1%80%D0%BE%D0%BB%D1%8F
4. “Хмарні сховища” [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/%D0%A5%D0%BC%D0%B0%D1%80%D0%BD%D1%96_%D1%81%D1%85%D0%BE%D0%B2%D0%B8%D1%89%D0%B0
5. “10 кращих програм для зберігання паролів” [Електронний ресурс] – Режим доступу: https://root-nation.com/ua/soft-ua/ua-10-krashhix-program-dlya-zberigannya-paroliv/
6. Лутц, М. Программирование на Python, I том / М. Лутц. - СПб.: Символ-плюс, 2015. - 992 с.
7. “PyQt” [Електронний ресурс] – Режим доступу: https://riverbankcomputing.com/software/pyqt/intro
8. Компактная встраиваемая реляционная база данных SQLite [Електронний ресурс] – Режим доступу: sqlite.org
9. Голицына Базы данных / Голицына, О.Л. и. - М.: Форум; Инфра-М, 2015. - 399 с.
10. “Язык UML” [Електронний ресурс] – Режим доступу: http://b-c-group.ru/?page_id=107

11. “Описание вариантов использования” [Електронний ресурс] – Режим доступу: http://www.nundesign.com/st/uml_doc/uml-elements.html
12. “RFC 7617” [Електронний ресурс] – Режим доступу: https://tools.ietf.org/html/rfc7617
13. “RestAPITutorial” [Електронний ресурс]. –Режим доступу:https://restfulapi.net/rest-architectural-constraints
14. Advanced Encryption Standard [Електронний ресурс]: (Стаття) / wikipedia.org. –Електрон. дан. (1 файл). –2017. – Режим доступу: https://en.wikipedia.org/wiki/Advanced_Encryption_Standard.
15. Кузьмин, Т. В. Криптографические методы защиты информации: моногр. / Т.В. Кузьмин. - Москва: Огни, 2013. - 192 с.
16. JSON [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/JSON.

ДОДАТОК А

Клієнт-серверне програмне забезпечення для шифрування та зберігання даних для входу на web-ресурси на базі операційної системи Linux.

Специфікація

_____

Аркушів 2

2020

Позначення	Найменування	Примітки
Документація		
Комплекс		
Компоненти		

ДОДАТОК Б

Клієнт-серверне програмне забезпечення для шифрування та зберігання даних для входу на web-ресурси на базі операційної системи Linux.

Текст програми

Аркушів 11

2020

Server.py

```
from core.CloudStorage import CloudStorage
from core.LogsController import LogsController
from core.UsersController import UsersController
from core.ShareController import ShareController
from core.NotesController import NotesController
from flask import Flask, abort
from flask import request, jsonify
from flask_httpauth import HTTPBasicAuth
from werkzeug.security import check_password_hash

app = Flask(__name__)
auth = HTTPBasicAuth()
logging = LogsController()

@auth.verify_password
def verify_password(username, password):
    try:
        userDB = UsersController()
        user = userDB.get_user(username)
        return check_password_hash(user.password, password)
    except:
        pass
    return False
# -----SINGLE USER ZONE----------
@app.route('/login')
@auth.login_required
def login():
    logging.add_log(str(auth.username() + " logged in"))
    return jsonify("200")


@app.route('/get_db')
@auth.login_required
def get_db():
```

```
    try:
        nc = NotesController()
        logging.add_log(str(auth.username() + " requested db"))
        return jsonify(nc.list_notes(str(auth.username())))
    except:
        return abort(501)
@app.route('/put_db_item', methods=['POST'])
@auth.login_required
def put_db():
    nc = NotesController()
    nc.add_note(str(auth.username()), request.form['data'], request.form['description'])
    logging.add_log(str(auth.username() + " appended db"))
    return jsonify("200")
@app.route('/del_db_item', methods=['POST'])
@auth.login_required
def del_db():
    nc = NotesController()
    nc.delete_note(str(auth.username()), request.form['id'])
    logging.add_log(str(auth.username() + " deleted db item"))
    return jsonify("200")
@app.route('/change_pass', methods=['GET'])
@auth.login_required
def change_pass():
    data = str(request.data.decode())
    try:
        userDB = UsersController()
        if userDB.change_password(str(auth.username()), data) == "200":
            logging.add_log(str(auth.username() + " changed password"))
            return jsonify("200")
        else:
            return jsonify("Bad pass!")
    except:
        return abort(501)
# ----------ADMIN ZONE----------
@app.route('/register_user', methods=['POST'])
```

```
@auth.login_required
def register_user():
    try:
        userDB = UsersController()
        if userDB.check_priveleges(str(auth.username())):
            userDB.new_user(request.form['username'], request.form['password'])
            logging.add_log(str(auth.username() + " added new user '" + request.form['username'] + "'"))
            return jsonify("200")
        return abort(401)
    except:
        return abort(501)
@app.route('/delete_user', methods=['POST'])
@auth.login_required
def delete_user():
    try:
        userDB = UsersController()
        notesDB = NotesController()
        if userDB.check_priveleges(str(auth.username())):
            notesDB.delete_all_notes(str(request.form['username']))
            userDB.delete_user(str(request.form['username']))
            logging.add_log(str(auth.username() + " deleted user '" + request.form['username'] + "'"))
            return jsonify("200")
        return abort(401)
    except:
        return abort(501)
@app.route('/list_users')
@auth.login_required
def list_users():
    try:
        userDB = UsersController()
        if userDB.check_priveleges(str(auth.username())):
            userDB = UsersController()
            list = userDB.list_all()
            logging.add_log(str(auth.username() + " requested user list"))
            return jsonify(list)
```

```
            return abort(401)
        except:
            return abort(501)
    @app.route('/give_admin', methods=['POST'])
    @auth.login_required
    def make_admin_user():
        try:
            userDB = UsersController()
            if userDB.check_priveleges(str(auth.username())):
                userDB.update_admin(str(request.form['username']))
                logging.add_log(str(auth.username() + " added admin rights to '" + request.form['username'] + "'"))
                return jsonify("200")
            return abort(401)
        except:
            return abort(501)


    @app.route('/take_admin', methods=['POST'])
    @auth.login_required
    def unmake_admin_user():
        try:
            userDB = UsersController()
            if userDB.check_priveleges(str(auth.username())):
                if str(request.form['username']) != 'admin':
                    userDB.update_admin(str(request.form['username']), 0)
                    logging.add_log(str(auth.username() + " removed admin rights from '" + request.form['username'] +
"'"))
                    return jsonify("200")
            return abort(401)
        except:
            return abort(501)


    @app.route('/log')
    @auth.login_required
```

```
def log():
    try:
        userDB = UsersController()
        if userDB.check_priveleges(str(auth.username())):
            return jsonify(logging.get_logs())
        return abort(401)
    except:
        return abort(501)
# -----------SHARING ENDPOINTS------------
@app.route('/new_keypair')
@auth.login_required
def get_new_keypair():
    try:
        shareDB = ShareController()
        shareDB.clear_shares(str(auth.username()))
        shareDB.close_connection()
        userDB = UsersController()
        logging.add_log(str(auth.username() + " requested new keypair"))
        return jsonify(userDB.generate_new_keypair(str(auth.username())))
    except:
        return abort(501)


@app.route('/share', methods=['POST'])
@auth.login_required
def share():
    try:
        shareDB = ShareController()
        logging.add_log(str(auth.username() + " shared note with '" + request.form['username'] + "'"))
        return jsonify(
            shareDB.share(str(auth.username()), request.form['username'], request.form['data'],
                          request.form['description']))

    except:
        return abort(501)
```

```
@app.route('/my_shares')
@auth.login_required
def my_shares():
    try:
        shareDB = ShareController()
        list = shareDB.list_shares(str(auth.username()))
        logging.add_log(str(auth.username() + " requested his shares"))
        return jsonify(list)
    except:
        return abort(501)



@app.route('/delete_share', methods=['POST'])
@auth.login_required
def del_share():
    try:
        shareDB = ShareController()
        shareDB.delete_share(request.form['id'], str(auth.username()))
        logging.add_log(str(auth.username() + " deleted shares"))
        return jsonify("200")
    except:
        return abort(501)
# -----------CLOUD STORAGE------------
@app.route('/add_file', methods=['POST'])
@auth.login_required
def add_file():
    cs = CloudStorage("data")
    cs.add_file(request.form['data'],str(auth.username()),request.form['description'])
    logging.add_log(str(auth.username() + " added file to cloud"))
    return jsonify("200")
@app.route('/my_files')
@auth.login_required
def my_files():
```

```
    try:
        cs = CloudStorage("data")
        list = cs.list_files(str(auth.username()))
        logging.add_log(str(auth.username() + " requested his file list"))
        return jsonify(list)
    except:
        return abort(501)

@app.route('/get_file', methods=['POST'])
@auth.login_required
def get_file():
    try:
        cs = CloudStorage("data")
        file = cs.get_file(str(auth.username()),request.form['id'])
        logging.add_log(str(auth.username() + " pulled file"))
        return jsonify(file)
    except:
        return abort(501)

@app.route('/delete_file', methods=['POST'])
@auth.login_required
def del_file():
    try:
        cs = CloudStorage("data")
        cs.del_file(str(auth.username()),request.form['id'])
        logging.add_log(str(auth.username() + " deleted file"))
        return jsonify("200")
    except:
        return abort(501)

if __name__ == '__main__':
    app.run("0.0.0.0",port=5000, ssl_context=('cert.pem', 'key.pem'))
```

Share.py

```
from core.DB import DB
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric import padding
import base64

class ShareController(DB):

    def __init__(self):
        super().__init__()
        create_table_sql = """ CREATE TABLE IF NOT EXISTS shares (
                                        id integer primary key, user_id integer, from_username text, data text, description
text, FOREIGN KEY (user_id) REFERENCES users(id) ); """
        self.select(create_table_sql)

    def share(self, from_user, to_user, data,description):
        sql = """SELECT pubkey from users where id=(SELECT id from users WHERE username=?)"""

        try:
            pubkey = self.select_vals(sql, (to_user,))[0][0]
            if pubkey == "None":
                return "No keypair yet!"
        except:
            return "No keypair yet!"

        with open("/tmp/" + to_user + ".pem", 'wb') as f:
            f.write(pubkey)

        with open("/tmp/" + to_user + ".pem", "rb") as key_file:
            public_key = serialization.load_pem_public_key(
                key_file.read(),
                backend=default_backend()
            )
```

```
            encrypted = public_key.encrypt(
                data.encode(),
                padding.OAEP(
                    mgf=padding.MGF1(algorithm=hashes.SHA256()),
                    algorithm=hashes.SHA256(),
                    label=None
                )
            )
            sql = """INSERT INTO shares (user_id,from_username,data,description) VALUES((SELECT id from users WHERE username=?),?,?,?)"""
            self.insert(sql, (to_user, from_user, encrypted,description))
            return "200"

        def list_shares(self, username):
            sql = """SELECT * from shares WHERE user_id=(SELECT id from users WHERE username=?)"""
            list =  self.select_vals(sql, (username,))
            result = {}
            for i, val in enumerate(list):
                result[i] = {"id":str(val[0]), "username":str(val[1]), "from":str(val[2]), "data":str(base64.b64encode(val[3]).decode()), "description":str(val[4])}
            return result

        def delete_share(self,id,username):
            sql= """DELETE FROM shares WHERE id=? AND user_id=(SELECT id from users WHERE username=?)"""
            self.insert(sql,(id,username))

        def clear_shares(self, username):
            sql = """DELETE FROM shares where user_id=(SELECT id from users WHERE username=?)"""
            self.insert(sql, (username,))
```

MyNotes.py (частина)

```
def setup(self):


    self.row_to_id = {}
    self.row_to_data = {}
    self.row_to_description = {}
    list = json.loads(self.db)
    i = 0
    self.listWidget.clear()
    self.listWidget.setCurrentRow(0)
    for key in list:
        self.listWidget.addItem(list[key]['description'])
        self.row_to_id[i] = int(key)
        self.row_to_data[i] = list[key]['data']
        self.row_to_description[i] = list[key]['description']
        i += 1



def delete_item(self):
    self.label.setText("")
    id = self.row_to_id[self.listWidget.currentRow()]
    self.notes_api.del_database_item(self.user, id)
    self.db = self.notes_api.get_database(self.user)
    self.setup()

def decrypt_sync(self):
    self.textBrowser.setText("")
    try:
        self.label.setText("")
        cryptor = AESCipher(self.getText("Master key:", False))
        enc = cryptor.decrypt(self.row_to_data[self.listWidget.currentRow()])
        self.textBrowser.setText(enc)
        if enc != "":
            self.label.setText("Successful decrypting")
        else:
```

```
            self.label.setText("Error decrypting")
        except:
            self.label.setText("Error decrypting")

    def new_note(self):
        try:
            text = self.plainTextEdit.toPlainText()
            if (text != ""):
                key = self.getText("Master key:", False)
                key_repeat = self.getText("Repeat master key:", False)
                if key == key_repeat:
                    cryptor = AESCipher(key)
                    enc = cryptor.encrypt(text)
                    self.notes_api.put_database_item(self.user, enc, self.getText("Description:"))
                    self.db = self.notes_api.get_database(self.user)
                    self.plainTextEdit.setPlainText("")
                    self.setup()
                    return None
        except:
            pass
        self.label.setText("Error")

    def new_keypair(self):
        try:
            self.share_api.new_keypair(self.user)
            self.label.setText("Keypair regenerated")
            return None
        except:
            pass
        self.label.setText("Error generating keypair")

    def share_note(self):
        self.label.setText("")
        try:
            cryptor = AESCipher(self.getText("Master key:", False))
```

```
            enc = cryptor.decrypt(self.row_to_data[self.listWidget.currentRow()])
            self.share_sig.emit(self.user, self.share_api, enc, self.row_to_description[self.listWidget.currentRow()])
            return None
        except:
            pass
        self.label.setText("Error decrypting")

    def my_shares(self):
        self.my_shares_sig.emit(self.user, self.share_api)

    def my_files(self):
            self.files_sig.emit(self.user)

    def settings(self):
        self.settings_sig.emit(self.user)

    def getText(self, request, echo=True, winName="Input"):
        property = QLineEdit.Password
        if echo:
            property = QLineEdit.Normal
        text, okPressed = QInputDialog.getText(self, winName, request, property, "")
        if okPressed and text != '':
            return text
```

ДОДАТОК В

Клієнт-серверне програмне забезпечення для шифрування та зберігання даних для входу на web-ресурси на базі операційної системи Linux.

Клієнт для взаємодії за хмарним сховищем конфіденційної інформації

Опис програмного модуля

_____

Аркушів 7

2020

-2-

Анотація

Розроблений клієнт призначений для безпечного зберігання та передавання конфіденційних даних, а саме – даних для входу.

Як вхідні дані клієнт отримує текст введений користувачем та майстер ключ.

Як результат клієнт шифрує дані за допомогою шифру AES та передає їх на сервер.

Як вхідні дані клієнт отримує унікальний номер запису, ключ та ім’я користувача.

Як результат клієнт розшифровує дані за допомогою шифру AES та передає їх на сервер разом із іменем користувача, якому вони призначені.

Як вхідні дані клієнт отримує унікальний номер запису та

Як результат клієнт розшифровує дані за допомогою шифру AES та виводить їх на екран.

Мова програмної реалізації —Python 3.

Середовище розробки —JetBrains PyCharm.

Зміст

-4-

1. Загальні відомості

Клієнт складається с модулю «client.py», який є точкою входу та керує усім графічними та логічними інтерфейсами програми. Мова програмної реалізації —Python Середовище розробки — JetBrains PyCharm.

Модуль працює з інформацією, яку вводить користувач, обробляє події на формі, формує запроси до сервера та обробляє серверні відповіді.

-5-

2. Функціональне призначення клієнту

Функціональним призначенням генератора є графічне представлення відповідей серверу та автоматизація процесів шифрування даних за допомогою синхронної та асинхронної криптографії.

Клієнт за допомогою звернення до серверу та методів модулів обробляє користувацькі події. Він звертається до наступних серверних методів:

- /login – basic аутентифікація
- /get_db – отримання записів поточного користувача
- /put_db_item – додати запис
- /del_db_item – видалити запис
- /change_pass – зміна паролю поточного користувача
- /register_user – реєстрація нового користувача
- /delete_user – видалити користувача
- /list_users – список зареєстрованих користувачів системи
- /give_admin – надати права адміністратора для користувача
- /take_admin – видалити права адміністратора у користувача
- /new_keypair генерування нової пари ключів
- /share поділитися записом із користувачем
- /my_shares список і зміст (зашифрований) записів, якими поділилися із поточним користувачем
- /delete_share – видалити запис із списку записів, якими поділилися із поточним користувачем
- /clear_shares – очистити список записів, якими поділилися із поточним користувачем
- /log – лог записи серверу

-6-

3. ВЗАЄМОДІЯ КЛІЄНТУ З ІНШИМИ КОМПОНЕНТАМИ СИСТЕМИ

Клієнт взаємодіє з модулями графічних форм, модулями реалізації криптографічних алгоритмів. Ці модулі містяться у папках gui та core.

4. Вхідні та вихідні дані

Як вхідні дані клієнт отримує текст введений користувачем та майстер ключ.

Як результат клієнт шифрує дані за допомогою шифру AES та передає їх на сервер.

Як вхідні дані клієнт отримує унікальний номер запису, ключ та ім’я користувача.

Як результат клієнт розшифровує дані за допомогою шифру AES та передає їх на сервер разом із іменем користувача, якому вони призначені.

Як вхідні дані клієнт отримує унікальний номер запису та

Як результат клієнт розшифровує дані за допомогою шифру AES та виводить їх на екран.